www.cbmstuff.com

SuperCard Pro

Software Developer's Kit

Manual v1.7

Release Date: December 23, 2013
Last Revision: December 7, 2015

Throughout this manual, SuperCard Pro board will be referred to as just 'SCP'. This manual describes the requirements to control the SCP using the USB port and serial port interfaces.

Hardware Information

The SCP contains the following hardware:

CPU: PIC24HJ256GP210A @ 160MHz (40 MIPS)
RAM: 512K Static RAM – 55ns
BUFFERS: 64mA/48mA drive, bi-directional
SD-MEDIA: micro-sd flip up cage
POWER: USB or external 3.5” floppy power interface, 100mA max
SERIAL: dual serial (300 baud through 1.25 megabaud)
USB: FTDI FT240-X, full-speed, parallel interface w/4K buffer
FLOPPY: 34 pin standard interface, with PIN 33 programmable
LEDS: 4 total, 2 green, 1 red, 1 blue
IEC: 4 pin expansion port for CBM IEC connector board

The SCP uses a standard Microchip programming interface port, so it is programmable using the Pickit 3, MPLAB ICD2/3, MeLabs USB programmers, etc. The SCP has a boot loader so that flashing can be done through the USB interface. If the PIC programming interface is used then the boot loader will be erased and the board will no longer have its original functionality. The firmware for SCP is closed source, and is not available as an executable or source code. So, only cbmstuff.com can re-flash the SCP to its original code.

See www.cbmstuff.com's forum area for information on updating the firmware.

Communicating with SCP

The SCP hardware uses data packets to control its functions. Commands can be sent to the SCP using either the USB port or either of the two serial ports. Note that port labeled 'RS232-2' is a half-duplex interface only. Port 'RS232-1' is setup as a half-duplex interface by default, but can be changed with solder-bridge jumpers to be a full duplex interface, having individual transmit (Tx) and receive (Rx) lines.

The FTDI USB FIFO chip used on SCP is simple to use. You open a .dll using C++, C#, Visual Basic, or .NET and make library calls to access the FTDI chip directly. Alternatively, you could force the USB driver to "Load VCP" which makes the USB interface appear as a standard Windows COM port. The COM port can then be opened and used just like a normal serial port. The baud rate is unused in this mode (12Mbps is the maximum speed).

For detailed information on programming the FTDI FT240-X chip, along with code examples, please see:

http://www.ftdichip.com/Support/SoftwareExamples/CodeExamples.htm

The software included with SCP was written in Visual Basic 6, which is probably the slowest method of communicating with the hardware and demonstrates that you don't need a fast device to control the SCP.

To control the SCP, you need to do the following:

Open the .dll or COM port
Build the packet
Send the packet
Get the response packet

That's it! Pretty simple!

Note that you can NOT build and send the packet on the fly. You must build the packet first and then send all of the data for that packet at one time. The USB communications end after 1ms of no bus activity, so it is possible that making some calculation could take longer and thus you will end up with a response code of *pr_Timeout*.

Also note that the SCP board should be plugged into a USB port directly, not a hub. There are some known issues with USB hubs and FTDI USB chips. This has gotten better in recent times, but the issue exists with certain types of hubs.

SCP Function Information

There are numerous functions that SCP can perform. Everything from selecting a drive to writing flux level data is achieved by sending packets of data via the USB or serial ports.

Packet Structure

All packets start with the command, followed by the payload length, the payload data itself, and an 8 bit checksum. The checksum is calculated with a starting value of 0x4A (ASCII 'J') and adding every byte in the packet together, up to, but not including the checksum itself. If the payload length is 0x00, then no payload data is present.

Examples below:

Command	Payload Length	Payload	Checksum
CMD_SELA	0x00		0x4A + CMD_SELA
CMD_STEPTO	0x01	Track #	0x4A + CMD_STEPTO + Track #
CMD_READFLUX	0x04	Length	0x4A + CMD_READFLUX + Length bytes

Response Packet returned = Command + Response code

To select drive A, you would send a 3 byte packet to the SCP: 0x80, 0x00, 0xCA. Some time after the packet is sent, a response packet is returned. The response packet consists of two bytes - the original command byte (0x80 in this case), and a single byte response code. The response packet will not be returned until the original packet's function has completed. Some functions could take many milliseconds to complete (like stepping of the head, seeking to track 0, reading data, etc.), while other functions will complete in just a few microseconds. Some packet functions are asynchronous, and therefore it is possible that a response packet returned will not match the function for the last packet that was sent. Make sure you compare the command byte in the packet you sent to the command byte in the response packet!

Most all commands require that the drive be selected prior to issuing a command. For example, in order to step the head the drive would need to be selected. Some floppy drives also require the motor to be enabled before stepping the head. This is completely drive dependent, but it is good practice to make sure the motor is spinning anyways.

Please note that some commands can only be used with one drive at a time. All cases of single drive used are noted in the description for the command.

Response Codes

The following is the list of possible response codes and their definitions:

```
pr_Unused = 0x00            ; not used (to disallow NULL responses)
pr_BadCommand = 0x01        ; bad command
pr_CommandErr = 0x02        ; command error (bad structure, etc.)
pr_Checksum = 0x03          ; packet checksum failed
pr_Timeout = 0x04           ; USB timeout
pr_NoTrk0 = 0x05            ; track zero never found (possibly no drive online)
pr_NoDriveSel = 0x06        ; no drive selected
pr_NoMotorSel = 0x07        ; motor not enabled (required for read or write)
pr_NotReady = 0x08          ; drive not ready (disk change is high)
pr_NoIndex   = 0x09         ; no index pulse detected
pr_ZeroRevs = 0x0A          ; zero revolutions chosen
pr_ReadTooLong = 0x0B       ; read data was more than RAM would hold
pr_BadLength = 0x0C         ; invalid length value
pr_Reserved1 = 0x0D         ; reserved for future use
pr_BoundaryOdd = 0x0E       ; location boundary is odd
pr_WPEnabled = 0x0F         ; disk is write protected
pr_BadRAM = 0x10            ; RAM test failed
pr_NoDisk = 0x11            ; no disk in drive
pr_BadBaud = 0x12           ; bad baud rate selected
pr_BadCmdOnPort = 0x13      ; command is not available for this type of port
pr_Ok = 0x4F                ; packet good (letter 'O' for OK)
```

The following is an example of the steps needed to select Drive A, turn on the motor for Drive A, and Seek to track 0. After each packet is sent, the response packet needs to be checked.

Select Drive A

Send: 0x80,0x00,0xCA
Get response packet: should be 0x80, 0x4F

Turn on Drive A's motor

Send: 0x84,0x00,0xCE
Get response packet: should be 0x84, 0x4F

Seek track 0

Send: 0x88,0x00,0xD2
Get response packet: should be 0x88, 0x4F

Packet Information

The following is the list of packets. Packet examples show the command value (in hex) followed by the definition, payload length, payload (if any), and checksum of the entire packet. A response packet is returned.

.b = byte, .w = word, .l = long

Note: Drive A = Drive 0, Drive B = Drive 1

0x80 - select drive A (Command.b, Payload Length.b, Checksum.b)
0x80 0x00 0xCA

This packet enables Drive A's select line.

0x81 - select drive B (Command.b, Payload Length.b, Checksum.b)
0x81 0x00 0xCB

This packet enables Drive B's select line.

0x82 - deselect drive A (Command.b, Payload Length.b, Checksum.b)
0x82 0x00 0xCC

This packet disables Drive A's select line.

0x83 - deselect drive B (Command.b, Payload Length.b, Checksum.b)
0x83 0x00 0xCD

This packet disables Drive B's select line.

0x84 – enable motor drive A (Command.b, Payload Length.b, Checksum.b)
0x84 0x00 0xCE

This packet turns on Drive A's motor.

0x85 - enable motor drive B (Command.b, Payload Length.b, Checksum.b)

0x85	0x00	0xCF

This packet turns on Drive B's motor.

0x86 – disable motor drive A (Command.b, Payload Length.b, Checksum.b)

0x86	0x00	0xD0

This packet turns off Drive A's motor.

0x87 – disable motor drive B (Command.b, Payload Length.b, Checksum.b)

0x87	0x00	0xD1

This packet turns off Drive B's motor.

0x88 - seek track 0 (Command.b, Payload Length.b, Checksum.b)

0x88	0x00	0xD2

This packet causes the currently selected drive to step the head to track 0 and stop.

0x89 - step to track (Command.b, Payload Length.b, Track.b, Checksum.b)

0x89	0x01	0x??	0x??

This packet steps the head to the track specified. The payload data byte contains the track to step to. ie. 0x89 0x01 0x23 0xF7 is command to step to track 35 (0x23).

NOTE: only one drive can be selected at a time using this command!

0x8A - step inwards (Command.b, Payload Length.b, Checksum.b)

0x8A	0x00	0xD4

This packet causes the currently selected drive to step once inwards to the next (higher) track.

0x8B - step outwards (Command.b, Payload Length.b, Checksum.b)

0x8B	0x00	0xD5

This packet causes the currently selected drive to step once outwards to the next (lower) track.

0x8C - select density (Command.b, Payload Length.b, Density.b, Checksum.b)

0x8C 0x01 0x?? 0x??

This packet set the density (high or low) of the currently selected drive. The payload data byte contains the density, either 0 (low density) or 1 (high density). The two possible packets for setting the density are:

0x8C 0x01 0x00 0xD7 is the packet used to set low density
0x8C 0x01 0x01 0xD8 is the packet used to set high density

NOTE: only one drive can be selected at a time using this routine!

0x8D - select side (Command.b, Payload Length.b, Side.b, Checksum.b)

0x8D 0x01 0x?? 0x??

This packet set the side (bottom or top) of the currently selected drive. The payload data byte contains the side to use. The two possible packets for setting the side are:

0x8D 0x01 0x00 0xD8 is the packet used to select side 0 (bottom)
0x8D 0x01 0x01 0xD9 is the packet used to select side 1 (top)

NOTE: only one drive can be selected at a time using this routine!

0x8E - get drive status (Command.b, Payload Length.b, Checksum.b)

0x8E 0x00 0xD8

This packet returns a word that has the current state of the drive control lines. The word is returned in big endian format immediately following the response packet.

Bit definition of word returned:

Bits 15-11, Reserved
Bit 10, Write Enable gate (0 = write enabled, 1 = write disabled)
Bit 9, Step direction (0 = inwards to track 40, 1 = outwards to track 0)
Bit 8, Density (0 = low density, 1 = high density)
Bit 7, Write protect (0 = write protected, 1 = not write protected)
Bit 6, Disk change/drive ready (0 = changed/drive not ready, 1 = no change/drive ready)
Bit 5, Track 0 (0 = on track 0, 1 = not on track 0)
Bit 4, Side (0 = side 1, top, 1 = side 0, bottom)
Bit 3, Drive B motor (0 = on, 1 = off)
Bit 2, Drive A motor (0 = on, 1 = off)
Bit 1, Drive select B (0 = selected, 1 = not selected)
Bit 0, Drive select A (0 = selected, 1 = not selected)

0x90 - get parameters (Command.b, Payload Length.b, Checksum.b)
0x90 0x00 0xDA

This routine returns 5 words (each in big endian) of parameters after first sending the return code of pr_OK.

Definition of returned words:

Word 1 - delay (in microseconds) after drive select (default - 1000)
Word 2 - delay (in microseconds) after step command (default 5000)
Word 3 - delay (in milliseconds) after motor on (default 750)
Word 4 - delay (in milliseconds) after seeking track 0 (default 15)
Word 5 - delay (in milliseconds) before automatically turning off all motors and deselecting all drives (default 20000)

0x91 - set parameters (Command.b, Payload Length.b, Parameters.w (x 5), Checksum.b)
0x91 0x0A 0x???? 0x??

The payload data contains the 5 words of parameters for the selected drive:

Word 1 - delay (in microseconds) after drive select (default - 1000)
Word 2 - delay (in microseconds) after step command (default 5000)
Word 3 - delay (in milliseconds) after motor on (default 750)
Word 4 - delay (in milliseconds) after seeking track 0 (default 15)
Word 5 - delay (in milliseconds) before automatically turning off all motors and deselecting all drives (default 20000). Setting this word to 0x0000 will disable the automatic shutoff routine. This routine disables all drives and motors after the period of time defined by this word.

0x92 - do RAM test (Command.b, Payload Length.b, Checksum.b)
0x92 0x00 0xDC

This routine does a test of the on-board 512K static RAM and returns either pr_OK on success or pr_BadRAM if the test fails.

0x93 - set PIN 33 of FDC (Command.b, Payload Length.b, State.b, Checksum.b)
0x93 0x01 0x?? 0x??

PIN 33 of the floppy drive connector is an output in copying mode (input in drive emulator mode). The state can be either 1 (high/on), or 0 (low/off). Making the pin high will enable any external device that can use the pin for control. The No-Flip (flippy) mod can be controlled using this command.

NOTE: outputting a high when PIN 33 has not been isolated from the floppy drive's ground path can result in damage to SCP's buffer chip!

0xA0 - read flux data (Command.b, Payload Length.b, # revolutions.b, flags.b, Checksum.b)

0xA0	0x02	(1-5)	0x??	0x??

This command reads a track of data at a flux level. The number of revolutions and flags are passed in. When the read has completed, pr_OK is returned when no error has occurred, otherwise a variety of error codes could be returned.

The flux data is stored starting at the beginning of the 512K RAM. If too much data is read, then an error condition of pr_ReadTooLong will be returned. After a read has been completed, the data stays in the RAM until it is changed by another read or command that makes use of the RAM.

Definition of flags:

bits 7-4 - reserved

bit 3 - ff_RPM360
0 = 300 RPM drive, 1 = 360 RPM drive

bit 2 - ff_Wipe
0 = no wipe before write, 1 = wipe track before write

bit 1 - ff_BitCellSize
0 = 16 bit cell size, 1 = 8 bit cell size

bit 0 - ff_Index
0 = immediate read, 1 = wait on index pulse before read

0xA1 - get info for last flux read (Command.b, Payload Length.b, Checksum.b)

0xA1	0x00	0xEB

This routine returns pr_OK, and then five sets of two longwords, representing the index time and number of bitcells for each revolution last read. All entries where no revolution was read will contain zeros (0x00000000). All longwords are in big endian format. Data is returned as follows:

```
Index time.l, # bitcells.l          ; revolution 1
Index time.l, # bitcells.l          ; revolution 2
Index time.l, # bitcells.l          ; revolution 3
Index time.l, # bitcells.l          ; revolution 4
Index time.l, # bitcells.l          ; revolution 5
```

This routine will typically be called immediately after calling command 0xA0 (read flux data) to get the information about the flux data that was read.

0xA2 - write flux data (Command.b, Payload Length.b, # of bitcells.l, flags.b, Checksum.b)

0xA2	0x05	0x????????	0x??	0x??

This command writes a track of data at a flux level. The number of bitcells to write and flags are passed in. When the write has completed, pr_OK is returned when no error has occurred, otherwise a variety of error codes could be returned.

The number of bitcells is the number of bitcell entries that will be written. Each bitcell entry is 16 bits in big endian format.

Definition of flags:

bits 7-4 - reserved

bit 3 - ff_RPM360
0 = 300 RPM drive, 1 = 360 RPM drive

bit 2 - ff_Wipe
0 = no wipe before write, 1 = wipe track before write

bit 1 - ff_BitCellSize
0 = 16 bit cell size, 1 = 8 bit cell size

bit 0 - ff_Index
0 = immediate write, 1 = wait on index pulse before write

0xA9 - send data from on-board RAM to USB port

(Command.b, Payload Length.b, RAM_Offset.l, TransferLength.l, Checksum.b)

0xA9	0x08	0x????????	0x????????	0x??

This command sends the data from on-board 512K static RAM to the USB port, using the offset into RAM and the transfer length.

The payload data contains the RAM offset and the transfer length, both as longwords in big endian format.

The RAM offset or transfer length can be an odd value.

After all data has been transferred, a response packet is then returned

Note: Unlike all other functions, the response packet is sent **after** all of the data has been received.

0xAA - get data from USB and store in RAM

(Command.b, Payload Length.b, RAM_Offset.l, TransferLength.l, Checksum.b)
0xAA 0x08 0x???????? 0x???????? 0x??

This command receives data from the USB port and stores it in the on-board 512K static RAM, using the offset into RAM and transfer length.

The payload data contains the RAM offset and the transfer length, both as longwords in big endian format.

The RAM offset or transfer length can be an odd value.

After all data has been received, the response packet is returned.

0xD0 - get info about SCP (Command.b, Payload Length.b, Checksum.b)
0xD0 0x00 0x1A

This command returns pr_OK, followed by two bytes containing the hardware version and firmware version:

Byte 1 - Hardware Ver<<4|Hardware Rev
Byte 2 - Firmware Ver<<4|Firmware Rev

0xD1 – set baud rate for RS232-1 (Command.b, Payload Length.b, BaudRate.b, Checksum.b)
0xD1 0x01 0x?? 0x??

This command sets the baud rate for the port. By default, RS232-1 is 250,000 baud (8,N,1).

The baud rate change is volatile, so resetting the SCP will restore the default baud rates.

After successfully changing the baud rate, this command returns pr_OK (or any error code) at the baud rate used **before** the requested change. Note: see command 0xD2 for the list of baud rate values and their associated baud rates.

To change the baud rate from default of 250,000 baud to 19,200 baud the packet sent would be:

0xD1, 0x01, 0x07, 0x23

0xD2 – set baud rate for RS232-2 (Command.b, Payload Length.b, BaudRate.b, Checksum.b)

0xD2	0x01	0x??	0x??

This command sets the baud rate for the port. By default, RS232-2 is 9600 baud (8,N,1).

The baud rate change is volatile, so resetting the SCP will restore the default baud rates.

After successfully changing the baud rate, this command returns pr_OK (or any error code) at the baud rate used **before** the requested change. See the list of baud rate values and their associated baud rates below:

Value	Baud rate
1	300 baud
2	600 baud
3	1200 baud
4	2400 baud
5	4800 baud
6	9600 baud
7	19200 baud
8	38400 baud
9	57600 baud
10	115200 baud
11	125000 baud
12	250000 baud
13	500000 baud
14	1250000 baud

List of SCP commands:

```
DoCMD_SELA          ; 0x80 - select drive A
DoCMD_SELB          ; 0x81 - select drive B
DoCMD_DSELA         ; 0x82 - deselect drive A
DoCMD_DSELB         ; 0x83 - deselect drive B
DoCMD_MTRAON        ; 0x84 - turn motor A on
DoCMD_MTRBON        ; 0x85 - turn motor B on
DoCMD_MTRAOFF       ; 0x86 - turn motor A off
DoCMD_MTRBOFF       ; 0x87 - turn motor B off
DoCMD_SEEK0         ; 0x88 - seek track 0
DoCMD_STEPTO        ; 0x89 - step to specified track
DoCMD_STEPIN        ; 0x8A - step towards inner (higher) track
DoCMD_STEPOUT       ; 0x8B - step towards outer (lower) track
DoCMD_SELDENS       ; 0x8C - select density
DoCMD_SIDE          ; 0x8D - select side
DoCMD_STATUS        ; 0x8E - get drive status
```

```
DoCMD_GETPARAMS          ; 0x90 - get parameters
DoCMD_SETPARAMS          ; 0x91 - set parameters
DoCMD_RAMTEST            ; 0x92 - do RAM test
DoCMD_SETPIN33           ; 0x93 - set pin 33 of floppy connector
DoCMD_READFLUX           ; 0xA0 - read flux level
DoCMD_GETFLUXINFO        ; 0xA1 - get info for last flux read
DoCMD_WRITEFLUX          ; 0xA2 - write flux level
  *DoCMD_READMFM         ; 0xA3 - read MFM level
  *DoCMD_GETMFMINFO      ; 0xA4 - get info for last MFM read
  *DoCMD_WRITEMFM        ; 0xA5 - write MFM level
  *DoCMD_READGCR         ; 0xA6 - read GCR level
  *DoCMD_GETGCRINFO      ; 0xA7 - get info for last GCR read
  *DoCMD_WRITEGCR        ; 0xA8 - write GCR level
DoCMD_SENDRAM_USB        ; 0xA9 - send data from buffer to USB
DoCMD_LOADRAM_USB        ; 0xAA - get data from USB and store in buffer
DoCMD_SENDRAM_232        ; 0xAB - send data from buffer to the serial port
DoCMD_LOADRAM_232        ; 0xAC - get data from the serial port and store in buffer

  *DoCMD_FLUX2CBMGCR     ; 0xB0 - convert flux data to CBM GCR data
  *DoCMD_GCR2CBMSECS     ; 0xB1 - convert CBM GCR data to raw sectors
  *DoCMD_READCBMSECS     ; 0xB2 - read CBM track and store in buffer as raw sectors
  *DoCMD_WRITECBMSECS    ; 0xB3 - write CBM track from raw sectors in buffer
  *DoCMD_READCBMDISK     ; 0xB4 – read CBM disk and store in buffer as raw sectors
  *DoCMD_WRITECBMDISK    ; 0xB5 - write CBM disk from raw sectors in buffer

  *DoCMD_OPENFILE        ; 0xC0 - open FAT16/32 file
  *DoCMD_CLOSEFILE       ; 0xC1 - close FAT16/32 file
  *DoCMD_READFILE        ; 0xC2 - read from open file
  *DoCMD_WRITEFILE       ; 0xC3 - write to open file
  *DoCMD_SEEKFILE        ; 0xC4 - seek to position in file
  *DoCMD_DELETEFILE      ; 0xC5 - delete file
  *DoCMD_FILEINFO        ; 0xC6 - get file info
  *DoCMD_DIR             ; 0xC7 - show directory
  *DoCMD_CHDIR           ; 0xC8 - change directory
  *DoCMD_MKDIR           ; 0xC9 - make new directory
  *DoCMD_RMDIR           ; 0xCA - remove directory

DoCMD_SCPInfo            ; 0xD0 - get info about SCP hardware/firmware
DoCMD_SETBAUD1           ; 0xD1 - sets the baud rate of port labeled RS232-1
DoCMD_SETBAUD2           ; 0xD2 - sets the baud rate of port labeled RS232-2
```

* Command not supported yet