SMSQ/E for Q40

Introduction

From the point of view of the hardware dependent features, SMSQ/E as implemented on the Q40 is very similar to other SMSQ implementations. The only significant differences are minor improvements.

The hard disk and floppy disk drivers can handle multiple disk formats, two floppy disk drives and four hard disk drives. Two IO cards can be used to provide up to 4 serial ports and 3 parallel printer ports.

Machine Type

The two standard functions to determine the machine type are, of course, supported.

MACHINE

The MACHINE function returns the machine type. This function returns 17 for the standard Q40.

PROCESSOR

The PROCESSOR function returns the 680x0 family member - 40 for the Q40.

Memory Protection

All production Q40s include a memory management unit but this is not yet fully used by SMSQ/E. The PROT_MEM procedure has, therefore, no effect in current versions and the supervisor mode access peeks and pokes do not have any different effect from there user mode cousins.

PROT_MEM

The PROT_MEM (level) procedure sets the level of the memory protection. This is ignored in current versions.

POKES POKES_W POKES_L

POKES (*address, value*) POKES_W (*address, value*) and POKES_L (*address, value*) are the "supervisor mode" equivalents of POKE, POKE_W and POKE_L. By operating in supervisor mode they enable data to be written to the QL IO hardware. Do not be surprised if your computer self-destructs when you use them.

PEEKS PEEKS_W PEEKS_L

PEEKS (*address*) PEEKS_W (*address*) and PEEKS_L (*address*) are the "supervisor mode" equivalents of PEEK, PEEK_W and PEEK_L. By operating in supervisor mode they enable data to be written to the QL IO hardware. Do not be surprised if your computer self-destructs when you use them.

Q40 Display

DISP_TYPE

The DISP_TYPE function is used to find the type of display. For the Q40, there are two values that may be returned.

0 Original ST QL emulator (this value is returned on QL based hardware).

1 16 bit colour mode.

```
ncol = 4                              Assume 4 colour display
if DISP_TYPE = 1: ncol = 65536  If it is 16 bit the are 65536 colours
```

DISP_INVERSE

The DISP_INVERSE (*0 or 1*) command is used to invert a monochome display. It has no effect on the Q40.

DISP_SIZE

DISP_SIZE (*xpixels, ylines*) is used to set the display size. When a size greater than 512x256 is specified, 16 bit colour mode is selected (not implemented in this version).

```
DISP_SIZE 1024,512          change to 1024x512 16 bit colour mode.
```

DISP_RATE

DISP_RATE (*frame rate, line rate*) is used to specify the frame and line scan rates. It has no effect on the Q40

DISP_BLANK

DISP_BLANK (*x blank, y blank*) sets the size of the blank area to the sides of and above and below the image. It has no effect on the Q40

Mouse driver

The mouse driver checks serial ports 4, 3, 2 and 1, in that order, looking for a Microsoft compatible two or three button mouse. If a mouse is used, it should be plugged into the highest port number available.

Because current serial mice for the PC have a much higher resolution (the pointer moves faster) than older mice, the original pointer interface scheme (accelerating slow mice) is no longer adequate. The new MOUSE_SPEED command is used to define the “acceleration”. The value specified for the MOUSE_SPEED is the same as the value that can be specified in the QPAC2 SYSDEF menu.

MOUSE_SPEED

MOUSE_SPEED (#*channel, speed, wake*) defines both a scaling for the mouse movement and an acceleration factor used for large movements.

Speed	Scaling	Acceleration
0	1/8	low
1	1/8	normal
2	1/4	low
3	1/4	normal
4	1/2	low
5	1/2	normal
6	1	low
7	1	normal
8	1	high
9	1	extreme

The optional *wake* defines how far the mouse will need to move before the pointer will appear (waking up the pointer) in a text input window.

The *channel* is optional, if the default channel is available, and is a console, no channel need be specified.

```
MOUSE_SPEED 2                    standard Microsoft mouse with low acceleration
MOUSE_SPEED #0,5,8                    cheap mouse with acceleration,
                                 pointer reluctant to wake up
```

Speeds 7 to 9 are the same as for previous versions. Speeds 0 to 6 are all slower than in previous versions. If a “low acceleration” speed is chosen, the pointer movement may be slightly viscous (this is an advantage in some applications). The default mouse speed is 7 (old mouse with normal acceleration). This default is overwritten by a configurable speed when QPAC2 is loaded.

The default wake speed is 3 which is fairly sensitive. This default is overwritten by a configurable speed when QPAC2 is loaded.

MOUSE_STUFF

MOUSE_STUFF (#*channel, string*) defines a 0, 1 or 2 character string to be stuffed into the keyboard queue when the centre (or left and right) buttons are pressed. This is usually used to send a Hotkey. If a Hotkey is required, the first character should be CHR$(255).

```
ERT HOT_THING (“.”, “Button_Pick”)     ALT . picks the button bar
MOUSE_STUFF CHR$(255) & ”.”                 The middle mouse button picks the button
bar
```

The default stuff string is CHR$(255) & “.”. This default is overwritten by a configurable Hotkey when QPAC2 is loaded.

The *channel* is optional, if the default channel is available, and is a console, no channel need be specified.

Serial (RS232) Ports on the Q40

The serial ports correspond to the standard IBM COM ports. Note that, unlike the PC BIOS SMSQ does not attempt to renumber the ports if it finds that one or more are missing

SER1	COM1	address $3F8
SER2	COM2	address $2F8
SER3	COM1/3	address $3E8
SER4	COM2/4	address $2E8

The baud rates correspond to the normal PC baud rates: standard rates up to 38400 baud and then 57600, 115200, 230400, 460800 and 921600 baud. Only 16550A/16450 compatible serial ports are supported (i.e. any IO card made in the past few years). The availability of rates above 115200 depends on whether the IO card supports these rates and whether the mechanism to produce these rates is recognised by the drivers.

```
BAUD 2,38400          . . . sets SER2 to 38400 baud
BAUD 57600            . . . sets SER1 to 57600 baud
```

All the SMSQ/E standard serial port control commands are avail.

Parallel Printer Ports

The parallel printer ports correspond to the standard IBM LPT ports.

PAR1	LPT1/2	address $378
PAR2	LPT2/3	address $278
PAR3	LPT1/2/3	address $3BC

The standard parallel port driver assumes that the parallel port is IEEE 1284 compatible (ECP) and it will normally operate in SPP FIFO mode. The port can also operate in original PC mode. There are three reasons for operating in original PC mode.

1. Some IO cards are not compatible, in ECP mode, with the Q40 interrupt system. If possible, this problem should be resolved by removing the IRQ7/IRQ5 jumper so that the card does not produce parallel port interrupts at all. It may, however, be necessary to set the jumpers on the card to SPP (original PC) mode.
2. Some printers may require a longer strobe pulse than is provided in FIFO mode.
3. It is PAR3 which is the LPT port at address $3BC. This is not an ECP port address.

PAR_PULSE

PAR_PULSE (*port, pulse length*) sets the notional strobe pulse length in ISA bus cycles. If the port is not specified, PAR1 is assumed. If the pulse length is zero, then the parallel printer port will operate in FIFO mode. If it is greater than 0, then the parallel printer port will operate in original PC mode.

```
PAR_PULSE 2,2         drive an old Epson printer on PAR2
PAR_PULSE 0           . . . set PAR 1 to FIFO mode
```

FIFO mode should be used if possible. The default value for PAR_PULSE is 0 if the IO card is configured for ECP mode or 1 if the IO card is configured for SPP mode.

PAR_WAIT

PAR_WAIT (*port, wait cycles*) sets the length of time that the parallel port drive will wait for the printer to be ready before it gives up and lets the Q40 do something else. This has no effect in FIFO mode, but in original PC mode it allows the buffer in the printer to be stuffed in bursts. The default value is 0. The larger the value, the higher the probability that a more than one byte of data can be sent on each interrupt, but the higher the load on the machine.

If the IO card does not provide IRQ7 and the machine is busy, PAR_PULSE with have a much greater effect than if IRQ7 is used and/or the machine is idle.

PAR_WAIT 2,20	*give the printer on PAR2 a high priority.*
PAR_WAIT 0	*. . . set PAR 1 use the minimum of processor time.*

For an Epson Stylus COLOR Pro printer, PAR_PAUSE 10 and PAR_PAUSE 50 improved the transfer speed by 30% on an idle machine: the rate was primarily determined by the printer. On a busy machine with no interrupts, PAR_PAUSE 10 improved the transfer speed by a factor of 3 and PAR_PAUSE 50 improved the transfer speed by a factor of 5. The speed of other tasks in the machine was reduced.

Q40 Hard Disks

IDE drives

The current IDE driver does not support removable drives.

WIN Drive Numbers and Name

ATA (IDE) drives are identified by the bus to which they are attached (primary or secondary), whether they are drive 0 or 1 on that bus (for historical reasons these are often called the master and slave drive although ATA compliant drives are neither master nor slaves: they are truly independent) and a partition on the drive.

Windows numbers its drives from C: as it finds them. This causes chaos if a removable media drive (or a normal drive in a rack) is used. (One of my PCs is obsessed by a phantom drive F: it thinks it is a 100kbyte CDROM).

SMSQ/E adopts a rather more cumbersome approach which is, however, much more precise. The initialisation code will attempt to find a file called "BOOT" on any partition on drive 0. WIN1 will be set to this partition. Thereafter, you must define your own WIN drives for any other drive and partition you wish to access.

This means that if, for example, you have a drive in a rack, the other drive numbers stay the same regardless of whether the drive is in or out when you boot the system.

SMSQ/E does not require the whole of a drive to be used for itself: the drive can be partitioned between different operating systems. Depending on the format of used by the other operating systems, SMSQ/E may be able to read or write these “foreign” partitions. Partitions are number from 0.

WIN_DRIVE

WIN_DRIVE (*drive, target, unit, partition*) is used to select a particular drive, unit and partition combination to be accessed using a particular WIN drive.

The "target" and "unit" notion comes from the SCSI bus terminology, the target is a physical device and the unit is a subdivision of that device. For IDE bus drives, there is only one unit per drive so the unit number is always zero and may be omitted. If the partition is omitted as well, then partition 0 (or the whole drive) is assumed.

Target	Bus	Drive
0	Primary	0 (Master)
1	Primary	1 (Slave)
2	Secondary	0 (Master)
3	Secondary	1 (Slave)

Issuing a WIN_DRIVE command for a particular drive will cause the drive map to be re-read the next time the disk is accessed. It can, therefore, be used to force the drivers to recognise a disk change.

```
WIN_DRIVE 2,0,1      WIN2 is drive 0 on the primary bus, partition 1
WIN_DRIVE 3,3        WIN3 is drive 1 on the secondary bus (whole drive or partition 0)
```

WIN_DRIVE$

WIN_DRIVE$ is a function which returns a string giving the target, unit and partition used by a particular WIN drive.

```
WIN_DRIVE 2,0,1      WIN2 is drive 0 on the primary bus, partition 1
WIN_DRIVE 3,3        WIN3 is drive 1 on the secondary bus (whole drive or partition 0)

PRINT WIN_DRIVE$(2)      Prints 0,0,1
PRINT WIN_DRIVE$(3)      Prints 3,0,0
```

WIN_USE

WIN_USE may be used to set the name of the WIN device. The name should be 3 characters long and in upper or lower case.

```
WIN_USE MDV          The WIN device is renamed MDV
WIN_USE win          The WIN device is restored to WIN
WIN_USE              The WIN device is restored to WIN
```

Formatting WIN Drives

If a drive is unformatted (or not recognisably formatted) you can format the whole drive as an SMSQ drive.

```
WIN_FORMAT 1         Allow WIN drives to be formatted
WIN_DRIVE 3,2        Set WIN3 to secondary drive 0, whole drive
FORMAT win3_Fred     FORMAT WIN3
WIN_FORMAT 0         Prevent WIN drives from being formatted
```

On the other hand, if you wish to share a drive between different operating systems, you can partition the drive by executing the MKPART utility before formatting.

WIN_FORMAT 1	*Allow WIN drives to be formatted*
WIN_DRIVE 3,2,1	*Set WIN3 to secondary drive 0, partition 1*
EW MKPART	*Partition drive, setting partition 1 to "QWA"*
FORMAT win3_Fred	*FORMAT WIN3*
WIN_FORMAT 0	*Prevent WIN drives from being formatted*

WIN Control Commands

There are a number of "odd" WIN device control commands.

WIN_WP

WIN_WP (*drive, 0 or 1*) is used to software write protect a WIN drive.

WIN_WP 1,1	*Set the "write protect" flag for the drive accessed by WIN1*
WIN_WP 1,0	*Clear the "write protect" flag for the drive accessed by WIN1*

WIN_START
WIN_STOP

The WIN_START (*drive*) and WIN_STOP (*drive, time)* commands may be used to start and stop a drive. If a time is given on the WIN_STOP command, the drive should not stop immediately: the time is the period without any disk accesses that that must elapse before the drive automatically enters standby mode. A zero time cancels the automatic standby timer.

WIN_STOP 2	*Stop the drive accessed by WIN2 now*
WIN_STOP 2,3	*Stop the drive accessed by WIN2 when there has been no access for 3 minutes*
WIN_STOP 2,0	*Do not stop the drive accessed by WIN2*
WIN_START 2	*Start the drive accessed by WIN2*

Note that all the operations that might be used to restart a drive (there is no "official" ATA command) are "vendor specific": on your particular drive, the drive may not start again until you try and read from (or write to) the drive, or it may never start again. You should also note that, on some drives, WIN_STOP *drive, time* will not only set the timer but stop the drive immediately as well.

As with any ATA command, these commands will work if they work, otherwise they will not work.

Q40 Floppy Disks

The Q40 will normally have one or two HD disk drives The SMSQ/E FLP driver can read or write QL5A, QL5B and MSDOS format diskettes. It can format QL5A (DD) and QL5B (HD) format diskettes.

Floppy Disk Driver Name

The default name of the floppy disk driver is FLP. The internal drive is FLP1. The external drive (if any) is FLP2.

FLP_USE

FLP_USE may be used to set the name of the FLP device. The name should be 3 characters long and in upper or lower case.

FLP_USE mdv	*The FLP device is renamed MDV*
FLP_USE FLP	*The FLP device is restored to FLP*
FLP_USE	*The FLP device is restored to FLP*

Formatting Diskettes

The SMSQ/E FLP driver will usually format a diskette to the highest density it can. The density may, however, be set using the FLP_DENSITY command or by adding a special code to the end of the medium name in the format command.

FLP_DENSITY

The SMSQ/E format routines will usually attempt to format a disk to the highest density possible for a medium. The FLP_DENSITY (*code*) is used to specify a particular recording density during format.

The density codes are "S" for single sided (double density), "D" for double density and "H" for high density.

FLP_DENSITY S	*Set the default format to single sided*
FLP_DENSITY H	*Set the default format to high density*
FLP_DENSITY	*Reset to automatic density selection*

The same code letters may be added (after a *) to the end of the medium name to force a particular density format. (For compatibility with older drivers, if the code letter is omitted after the *, single sided format is assumed.

FORMAT 'FLP1_Disk23'	*Format at highest density or as specified by FLP_DENSITY*
FORMAT 'FLP1_Disk24*'	*Format single sided*
FORMAT 'FLP1_Disk25*S'	*Format single sided*
FORMAT 'FLP1_Disk25*D'	*Format double sided, double density*

FLP_TRACK

The FLP_TRACK (*number of tracks*) is used to limit the number of tracks formatted.

FLP_TRACK 23	*Only format 23 tracks*

FLP Control Commands

FLP_SEC

FLP_SEC (*level*) was used to set the security level. The security of the data stored on the diskettes can be seriously compromised if you change diskettes while there are files open. The security level affects the amount of time the FLP driver spends maintaining the data on the diskette up to date with the internal copies of the data in memory. In principle, a lower level is more efficient, but more risky. With the increasing use of hard disks, the security level of the FLP has been fixed at level 2: the most secure. FLP_SEC is ignored.

FLP_START

The FLP_START (*ticks*) command specifies the number of ticks (1/50th of a second) that the FLP driver waits after starting the drive before writing to it. This allows the diskette to get up to speed before the write operation. The default value is 24, which is a wait of about 0.5 s. There should not be any reason to use this command.

FLP_STEP

The FLP_STEP (*drive, step*) command specifies the step rate for a particular drive. If the drive number is omitted, the step rate applies to both drives. The step rate will be adjusted downwards by the driver if there are repeated seek errors. The FLP_STEP command should not, therefore, be necessary.

```
FLP_STEP 2,15          Set FLP2 to 15 ms step rate
FLP_STEP 3             Set both drives to 3 ms step rate.
```

Sampled Sound System

The SMSQ/E sampled sound system for the Q40 assumes that a sampling rate of 20 kHz will always be used.

The system is based on a 2 byte wide queue. Sound generators should stuff pairs of bytes (left, right) in the queue. The queue is 200 kilobytes long which allows up to 5 seconds free running. A normal “boing” can be set up in a single operation.

The SMSQ/E sampled sound system provides four basic functions to add a single sample, to add an arbitrary number of samples, to stop the sound and to estimate the length of sound samples remaining in the queue.

The SMSQ/E sampled sound system should be accessed in supervisor mode (in principal, this will be a sound device driver) via the interrupt level 4 auto vector.

```
      move.l   $70,a3          interrupt level 4 auto vector
      move.l   -(a3),a2        address of sample sound system functions
      cmp.l    #'SSSS',-(a3)   SMSQ/E Sampled Sound System
      bne.s    oops
…     jsr      $04(a2)         add a sample
…     jsr      $08(a2)         set up to add multiple samples
…     jsr      $0c(a2)         notify that multiple samples have been added
…     jsr      $10(a2)         kill the sound
```

SSS_ADD1 ($04)

The sss_add1 call is used to add one sample to the sound queue. To limit the overheads, it does not save any registers.

```
D1   call byte     left hand sound level
D2   call byte     right hand sound level
A1   smashed
A3   call          pointer to ‘SSSS’ flag (see code above)
```

The sound level is a byte value between 0 and 255. The sound “zero” level is 128. This should be the last value written to the left and right hand sound queues.

This call does not have a standard error return. It returns status Z if the sample has not been added because the queue is full.

SSS_SETM ($08)

The sss_setm call sets up to add multiple samples to the sound queue.

```
A1   return        the pointer to the next free byte pair in the queue
A2   return        the pointer past the last free byte pair in the queue
A3   call          pointer to ‘SSSS’ flag (see code above)
```

The calling routine can fill the area from a1 to a2 with pairs of bytes. It does not, however, need to fill the whole of the area. When it has put samples into the queue, it should call SSS_ADDM to notify the sampled sound system.

SSS_ADDM ($0C)

The sss_addm call notifies that samples have been added to the sound queue.

A1	call	the updated pointer to the next free byte pair in the queue
A3	call	pointer to 'SSSS' flag (see code above)

```
        move.l  $70,a3                  interrupt level 4 auto vector
        move.l  -(a3),a2                address of sample sound system functions
        cmp.l   #'SSSS',-(a3)           SMSQ/E Sampled Sound System
        bne.s   oops

        jsr     sss_setm(a2)            set up
        bra.s   end_loop                note, a1 might be equal to a2
loop
        calculate next sample in d1.b, d2.b
        move.b  d1,(a1)+                add left sample
        move.b  d2,(a1)+                add right sample
end_loop
        cmp.l   a2,a1                   more samples to do?
        blt.s   loop

        jsr     sss_addm(a2)            notify sampled sound system
```

SSS_KILL ($10)

The sss_kill call stops the sound system and throws the queue away.

A3	call	pointer to 'SSSS' flag (see code above)

SSS_SAMPLE ($14)

The sss_sample call estimates the number of samples remaining in the queue. This figure should be divided by 400 to give the length of the sound in ticks or divided by 20000 to give the length of sound in seconds.

D0	return long	number of samples remaining in queue
A3	call	pointer to 'SSSS' flag (see code above)