NST's Audio Extension V2.0 ©2011-2014

(Manual V1.2.1, 2014.06.07.)

Features:

- 8580[1] SID support ("new" SID; chip not included)
- DigiBlaster playback emulation (8-bit D/A)
- Standard SID base addresses ($FD40, $FE80), $D400 option
- Standard plus/4 clock (886kHz), PAL C64 clock (985kHz)
- "SID Reset" button for "legacy C64" mode
- ”Configurable analog signal path”™
- 3.5mm audio jack
- ~C64 compatible joystick port
- AMIGA / ATARI ST mouse support
- And a lot more...

1 6581 compatible cards upon request

The hardware

There are two, significantly different generations of SID chips: 6581 and 8580 (the latter also known as 6582). These chips need different electrical setup. The first production run of SID cards supported 8580 exclusively. From the second production run, 8580 is default, and 6581 specific cards are available upon request. This is a static setup (in all, 8 components need to be physically different). Never attempt to use 8580 SIDs in 6581 specific cards (there's an almost absolute risk of damaging both the 8580 and the card). 6581 SIDs in 8580 specific cards are likely to play no sound at all, although there's no risk of damage for this scenario.

There's a mark for Pin 1 of the SID chip on the board ("Pin1" tag, next to the corner of the SID socket near the 3.5mm audio jack connector). **Inserting the SID chip with wrong orientation will damage both the SID chip and the card!**

The SID chip needs to be inserted the following way:

Pay attention that all pins slide into the the socket, no pins should bend or break. (Precaution might be especially important if some pins of the SID are already slightly crooked, most likely due to the chip being a pull-off ie. a used piece. Smoothing the pins using small tweezers prior to install may help here a lot. Don't apply unreasonably high force, try positioning the pins so that they'd go into the holes, and take care that none of the pins get bent or broken off while installing the chip in the socket).

WARNING! You should not power up the card without a SID chip installed!

There are 3 blue jumpers on the card (J1, J2, J3) determining how the sound signals of the SID and the TED will be routed. By default, all jumpers are in 1-2 position, like this:

In this setup, the sound output of the SID ends up at the A/V output of the plus/4 (ie. the "Video" socket), mixed together with the TEDs sound. (Note: with this setup the SID sound should also be available on the TV if you're using an RF connection). If headphones or an amp is connected to the 3.5mm headphone socket, the SIDs sound signal path will be cut at that point – ie. with headphones or an amp connected, the plus/4 A/V socket and optionally the RF out no longer plays the SIDs sound.

This is the default, recommended configuration.

Optionally, it is possible to do the opposite, ie. mix the TEDs sound to the SIDs sound on the card itself with both signals ending up at the cards 3.5mm audio socket. For that, move J2 and J3 to position ”2-3”, like this:

In that case, both the SIDs and TEDs sound appear, mixed, at the audio jack socket.
ATTENTION: J2 and J3 needs to be moved together. **Don't attempt to set the other, physically possible combinations!**

If J2 and J3 are set "2-3", like the above, then the two signal sources (the SID and the TED) can be set further "apart" using J1; that is, the SIDs sound will appear on one, and the TEDs at the other channel of the stereo signal. This setup would look like this:

In this mode, you can play sounds in stereo on the plus/4. (I wouldn't encourage anyone to do stereo that way, though; yet, the possibility is there.)

There's yet one more, red jumper on the card, on the 1-2 pins of the "AuIn" connector. Pin 2 of this header is basically the Ext Audio input of the SID chip. Pin 1 is the output of the emulated DigiBlaster audio playback. Pin 3 is GND. The jumper can be moved to 2-3 if DigiBlaster emulation is not needed. (Like this:)

(The D/A used here is "cheap" component that might carry in some noise even if not in use; this noise can also be eliminated this way, if significant).

This jumper can also be taken advantage of if the SIDs Ext Audio input is needed for some purpose. For that, the jumper has to be removed, and the audio signal be connected to pins 2 and 3 (pin2: signal, pin3: GND) of the header. Pay attention to the fact that this point of the card is **absolutely unprotected**. According to SID specs, the **maximum level** that can be fed to the **SIDs input is 1 Vpp**. The SID isn't a typically robust, forgiving device, you can make damages to the chip pretty easily, so **only use this possibility if you really know what you're doing!**

Messing around with analog audio signals generally leads to some loss of audio quality. All settings but the default one imposes some extra routing of analog sound signals. Consequently, the suggested configuration is the default one. For best performance, the SIDs sound should be taken from the audio jack, and the TEDs sound from the computers A/V socket.

There's a button on the card ("SRST", ie. Sid ReSeT). This button has two functions:

- The SID can be reset, especially if access to the SID in the $D400 – $D41F area had been enabled previously (and there's some unwanted sound due to programs overwriting this area).
- If the button is kept pressed while the Reset button of the plus/4 is released, "Legacy C64 mode" is activated (SID access at $D400 – $D41F is enabled, and the SID is clocked at 985kHz).

It's suggested that you use this "Legacy C64 mode" with connecting an amplifier or headphones to the audio jack socket of the card. In this case, a great number of plus/4 demos will play SID sound (and only SID sound) even if SID support has never been implemented in them at all. (This is due to the usual scenario of original C64 sound routines playing somewhere deep in these productions). The TEDs sound (that is, the "converted" sound in this case), as the audio is taken from the card's audio socket directly, will be absent.

There's a C64 compatible joystick port on the card. You can connect a C64 compatible joystick, a 1350/1351 (or compatible) mouse, or optionally an Amiga/Atari ST mouse here. C64 lightpens aren't supported; originally, these would be handled by a VIC-II, which is absent here (...and lightpens don't work with modern displays anyway).

The software

The SID can be accessed at $FD40 – $FD5F, which is the "default" address map. By default, the map is also mirrored at $FE80 – $FE9F (backwards compatibility to Csory's SID card), which can be disabled. Optionally, a mirror for write access of the SID registers can be enabled for the $D400 – $D41F address range ("Legacy C64 mode", disabled by default).

The original DigiBlaster extension for the Synergy SID card features an 8-bit A/D and D/A converter, basically, an early form of digital sound codec. NAE supports the DigiBlaster digital sound replay function just out of the box. (The A/D converter is not implemented). The D/A register is mapped to an unused address of the SID address map, $1E ($FD5E or $FE9E). (The A/D converter, if implemented, would map to address $1F). The output of the D/A, like in the original DigiBlaster design, is fed to the "Ext Audio" input of the SID, ie. the signal passes through the SID; as a consequence, the volume setting of the SID affects the volume level of the sample playback of the DigiBlaster, and the DigiBlasters sound can also be filtered using the SIDs programmable filters. (As another consequence, not even the DigiBlaster sample playback can be used without a SID chip installed.)

The card's own registers reside in the $FD80 – $FD8F range, which is originally assigned to the joystick port of the Synergy SID card (there, the single joystick input register is mirrored 16 times). For compatibility, the original behaviour can also be specified. (There's a bug in the first firmware version (a.k.a V1.0) of this compatibility mode of the card, see below.)

SID registers (summary):

Addr.	*I/O*	*B7*	*B6*	*B5*	*B4*	*B3*	*B2*	*B1*	*B0*
$FD40 $FE80	**Write only Channel 1**	Freq 7	Freq 6	Freq 5	Freq 4	Freq 3	Freq 2	Freq 1	Freq 0
$FD41 $FE81	**Write only Channel 1**	Freq 15	Freq 14	Freq 13	Freq 12	Freq 11	Freq 10	Freq 9	Freq 8
$FD42 $FE82	**Write only Channel 1**	PW 7	PW 6	PW 5	PW 4	PW 3	PW 2	PW 1	PW 0
$FD43 $FE83	**Write only Channel 1**	-	-	-	-	PW 11	PW 10	PW 9	PW 8
$FD44 $FE84	**Write only Channel 1**	Noise	Pulse	Sawtooth	Triangle	Test	RingMod	Sync	Gate
$FD45 $FE85	**Write only Channel 1**	Attack 3	Attack 2	Attack 1	Attack 0	Decay 3	Decay 2	Decay 1	Decay 0
$FD46 $FE86	**Write only Channel 1**	Sustain 3	Sustain 2	Sustain 1	Sustain 0	Release 3	Release 2	Release 1	Release 0
$FD47 $FE87	**Write only Channel 2**	Freq 7	Freq 6	Freq 5	Freq 4	Freq 3	Freq 2	Freq 1	Freq 0
$FD48 $FE88	**Write only Channel 2**	Freq 15	Freq 14	Freq 13	Freq 12	Freq 11	Freq 10	Freq 9	Freq 8
$FD49 $FE89	**Write only Channel 2**	PW 7	PW 6	PW 5	PW 4	PW 3	PW 2	PW 1	PW 0
$FD4A $FE8A	**Write only Channel 2**	-	-	-	-	PW 11	PW 10	PW 9	PW 8
$FD4B $FE8B	**Write only Channel 2**	Noise	Pulse	Sawtooth	Triangle	Test	RingMod	Sync	Gate
$FD4C $FE8C	**Write only Channel 2**	Attack 3	Attack 2	Attack 1	Attack 0	Decay 3	Decay 2	Decay 1	Decay 0
$FD4D $FE8D	**Write only Channel 2**	Sustain 3	Sustain 2	Sustain 1	Sustain 0	Release 3	Release 2	Release 1	Release 0
$FD4E $FE8E	**Write only Channel 3**	Freq 7	Freq 6	Freq 5	Freq 4	Freq 3	Freq 2	Freq 1	Freq 0
$FD4F $FE8F	**Write only Channel 3**	Freq 15	Freq 14	Freq 13	Freq 12	Freq 11	Freq 10	Freq 9	Freq 8
$FD50 $FE90	**Write only Channel 3**	PW 7	PW 6	PW 5	PW 4	PW 3	PW 2	PW 1	PW 0
$FD51 $FE91	**Write only Channel 3**	-	-	-	-	PW 11	PW 10	PW 9	PW 8
$FD52 $FE92	**Write only Channel 3**	Noise	Pulse	Sawtooth	Triangle	Test	RingMod	Sync	Gate
$FD53 $FE93	**Write only Channel 3**	Attack 3	Attack 2	Attack 1	Attack 0	Decay 3	Decay 2	Decay 1	Decay 0
$FD54 $FE94	**Write only Channel 3**	Sustain 3	Sustain 2	Sustain 1	Sustain 0	Release 3	Release 2	Release 1	Release 0
$FD55 $FE95	**Write only Filter**	-	-	-	-	-	FC2	FC1	FC0
$FD56 $FE96	**Write only Filter**	FC10	FC9	FC8	FC7	FC6	FC5	FC4	FC3
$FD57 $FE97	**Write only Filter**	Res 3	Res 2	Res 1	Res 0	Filt ExIn	Filt Ch3	Filt Ch2	Filt Ch1
$FD58 $FE98	**Write only Filter**	Ch3 Off	HP	BP	LP	Vol 3	Vol 2	Vol 1	Vol 0
$FD59 $FE99	**Read Only PotX**	PotX 7	PotX 6	PotX 5	PotX 4	PotX 3	PotX 2	PotX 1	PotX 0
$FD5A $FE9A	**Read Only PotY**	PotY 7	PotY 6	PotY 5	PotY 4	PotY 3	PotY 2	PotY 1	PotY 0
$FD5B $FE9B	**Read Only OSC3/Rnd**	Osc3 7	Osc3 6	Osc3 5	Osc3 4	Osc3 3	Osc3 2	Osc3 1	Osc3 0
$FD5C $FE9C	**Read Only ENV3**	Env3 7	Env3 6	Env3 5	Env3 4	Env3 3	Env3 2	Env3 1	Env3 0
$FD5D $FE9D	**N/A**	-	-	-	-	-	-	-	-
$FD5E $FE9E	**Write only DigiBlaster**	D/A B7	D/A B6	D/A B5	D/A B4	D/A B3	D/A B2	D/A B1	D/A B0
$FD5F $FE9F	**N/A**	-	-	-	-	-	-	-	-

SID registers (there are a lot of documents dealing with SID register and operation details around the *internet*, so this is only a summary):

$FD40/$FD41 (channel 1):
$FD47/$FD48 (channel 2):
$FD4E/$FD4F (channel 3):
Frequency of the channel (16 bit, write only)

Freq determines the frequency that the selected waveform of the channel is played at. The higher the number, the higher the pitch of the sound.

$FD42/$FD43 (channel 1):
$FD49/$FD4A (channel 2):
$FD50/$FD51 (channel 3):
Pulse width of the square waveform (proportion) (12 bit, write only)

PW 0..11 determine pulse width. Value is valid between $000 – $FFF (bits 12-15 are unused). $800 results in a 50% square wave. Only valid for square waveform.

$FD44 (channel 1):
$FD4B (channel 2):
$FD52 (channel 3):
Channel control register (write only)

B7: Noise bit: Select noise waveform
B6: Pulse bit: Select square waveform
B5: Sawtooth bit: Select sawtooth waveform
B4: Triangle bit: Select triangle waveform
B3: Test bit
B2: RingMod bit: Turn on ring modulation
B1: Sync bit: Turn on oscillator sync
B0: Gate bit: 1 starts attack phase of ADSR generator, 0 starts release

More than one of *B7 – B4* can be turned on at the same time (but not all combinations result in audible sound).

$FD45/$FD46 (channel 1):
$FD4C/$FD4D (channel 2):
$FD53/$FD54 (channel 3):
ADSR generator parameters (write only)

The registers hold four 4-bit parameters: attack, decay, sustain, release. Having turned the Gate bit on in the control register, the volume of the respective channel rises up to maximum by "attack" rate, then drops by "decay" rate until the volume level of "sustain" is reached. Resetting the Gate bit lets the volume level drop to 0 by "release" rate.

$FD55/$FD56:
Cutoff frequency of the programmable multimode filter (write only)

Bits B7-B3 of $FD55 are unused.

$FD57:
Filter control bits and resonance (write only)

B7 – B4: Filter resonance
B3: Filter ExIn: Route Ext Audio signal through the filter
B2: Filter Ch3: Route channel 3 through the filter
B1: Filter Ch2: Route channel 2 through the filter
B0: Filter Ch1: Route channel 1 through the filter

$FD58:
Filter mode and master volume (write only)

B7: Channel 3 off bit: If 1, channel 3 is muted
B6: HP bit: Select high pass filter
B5: BP bit: Select bandpass filter
B4: LP bit: Select low pass filter
B3 – B0: master volume

$FD59/$FD5A:
Paddle inputs (read only)

The input registers of the two single-slope A/D converters of the SID.

$FD5B:
Channel 3 wave value (read only)

Upper 8 bits of Channel 3's waveform generator .

$FD5C:
Channel 3 ADSR generator value (read only)

Current volume level of Channel 3 (ie. the current value of the volume counter of the ADSR generator of Channel 3)

$FD5D:
Unused

$FD5E:
DigiBlaster D/A (write only)

This is actually not a SID register. The D/A register of the DigiBlaster is mapped to this address. The output of this D/A is routed through the SID. Because of that, the DigiBlasters sound is only audible if the master volume of the SID is non-zero.

$FD5F:
Unused

The card's specific registers reside in the $FD80 – $FD8F range.

Register map (summary):

Addr.	*I/O*	*B7*	*B6*	*B5*	*B4*	*B3*	*B2*	*B1*	*B0*
$FD80	**Read only**	1	1	1	Joy FIRE	Joy RIGHT	Joy LEFT	Joy DOWN	Joy UP
$FD81	**Read only**	1	Joy „PotY”	Joy „PotX”	Joy FIRE	Joy RIGHT	Joy LEFT	Joy DOWN	Joy UP
$FD82	**Read only**	Mouse X b7	Mouse X b6	Mouse X b5	Mouse X b4	Mouse X b3	Mouse X b2	Mouse X b1	Mouse X b0
$FD83	**Read only**	Mouse Y b7	Mouse Y b6	Mouse Y b5	Mouse Y b4	Mouse Y b3	Mouse Y b2	Mouse Y b1	Mouse Y b0
$FD84	**N/A**								
$FD85	**N/A**								
$FD86	**N/A**								
$FD87	**N/A**								
$FD88	**Read only**	LastSID D7	LastSID D6	LastSID D5	LastSID D4	LastSID D3	LastSID D2	LastSID D1	LastSID D0
$FD89	**Read only**	LastSID R/W	0	0	LastSID A4	LastSID A3	LastSID A2	LastSID A1	LastSID A0
$FD8A	**N/A**								
$FD8B	**N/A**								
$FD8C	**N/A**								
$FD8D	**Write only**	Config Comm.7	Config Comm.6	Config Comm.5	Config Comm.4	Config Comm.3	Config Comm.2	Config Comm.1	Config Comm.0
$FD8E	**Read only**	PAL/ NTSC	0	Mouse type	Mouse mode	DigiBl. enable	$FE80 enable	$D400 enable	886kHz 985kHz
$FD8F	**Read only**	Card type	Version b2	Version b1	Version b0	Revision b3	Revision b2	Revision b1	Revision b0/SIDM

Detailed description of the registers:

$FD80:
Input register of the joystick port (read only)

Inputs are L-active (that is, inactive joystick inputs are 1, currently active inputs are 0). The register has two operating modes, set by the "Mouse mode" bit.
By default, the card is in "Analog mode" (1350/1351 compatible mode), and no joystick register bits remapping is done.
In "digital mode" (Amiga / Atari ST mouse), the register bits are remapped, so that programs don't have to take care about the type of mouse actually used. (The bits effectively behave like there was a 1351 mouse connected to the port).

B7, B6, B5: unused bits ("1" when read, similarly to the original Synergy SID card by Solder).
B4: Joystick FIRE button / 1351 LMB : 0: pressed, 1: released
B3: Joystick RIGHT: 0: active, 1: inactive
B2: Joystick LEFT: 0: active, 1: inactive
B1: Joystick DOWN / Amiga/Atari MMB: 0: active, 1: inactive
B0: Joystick UP / 1351 / Amiga/Atari RMB: 0: active, 1: inactive

$FD81:
Input register of the joystick port (read only)

Similar in function to $FD80, with the exception that it gives back the true state of the joystick port, regardless of "Mouse mode" (that is, no remapping is done with respect to this register, even if $FD80 is remapped).

B7: Unused bit (reads 1)
B6: "Mouse mode" is "digital": MMB, 0: pressed, 1: released; mode is "analog": unused, reads 1
B5: "Mouse mode" is "digital": RMB, 0: pressed, 1: released; mode is "analog": unused, reads 1
B4: Joystick FIRE button / 1351 / Amiga/Atari LMB : 0: pressed, 1: released
B3: Joystick RIGHT: 0: active, 1: inactive
B2: Joystick LEFT: 0: active, 1: inactive
B1: Joystick DOWN: 0: active, 1: inactive
B0: Joystick UP / 1351 RMB: 0: active, 1: inactive

In practice, RMB and MMB are only accessible in "digital" mouse mode. The function of some port pins of the joystick port differ in standard "analog" and "digital" mode (as do the layouts of standard C64 and Amiga / Atari ST joystick ports). In order to read RMB and MMB, control of the respective port pins need to be withdrawn from the SID (which is done likewise in "digital" mode).

$FD82:
Digital mouse X position (read only)

$FD83:
Digital mouse Y position (read only)

These are the X and Y counters of the Amiga / Atari ST mouse position. With the mouse moved from left to right / front to backwards, X / Y increases (and vice versa). These counters wrap around (that is, they hold the lowest 8 bits of the respective "absolute" mouse position value). This is an unusual scenario for standard mice ("intelligent" mice generally speak relative displacement, "bare" mice speak quadratures), however, from a programmers perspective, it's very similar to how a Commodore 1351 works.

In "digital" mouse mode, these registers also override the SIDs POTX/POTY registers; that is, they get mapped to $FD59 and $FD5A, respectively. Compared to the 1351, the Y position counter of the Amiga / Atari ST mouse counts to the opposite direction. For backwards compatibility, all bits of the remapped mirror of the Y register are inverted, so that this direction of the emulated 1351 would be correct. (Warning: firmware V1.0 didn't implement this inversion, thus, prior to V1.2, 1351 emulation from an Amiga / Atari ST mouse is incorrect).

$FD84,
$FD85,
$FD86,
$FD87:
Unused (reads 0)

$FD88:
Parameter of last SID register access (read only)

B7-B0: Written or read data of last SID access

$FD89:
Additional parameters of last SID register access(read only)

B7: R/W bit: 0: WRITE, 1: READ (the type of last SID access)
B4-B0: A4-A0: Address of SID register accessed

The latched parameters of the last SID CPU access cycle. These registers are accessible at any time, but there's no use of using them except for the case of the SID being clocked at 985kHz. In this mode the SID runs asynchronously to the plus/4, which means that its registers can be written to (which is buffered by hardware), but can't be read from (to put precisely: they can, but won't result in predictably accessible data for the CPU). However: this is a limitation of the system bus, not the SID card. The results of those SID accesses can be, and actually are latched by the card, and the latched values can then be read from the registers above. For that, the following method could be used:

```
LDA   $FD59     ← Read SID PotX, throw invalid result away (#1)
LDA   $FD88     ← Read the result of previous SID operation (#2)
```

In other words: perform the SID operation, throw the likely invalid result away, and read the actual result from $FD88.

There's a small chance of failure in the above method, however: reading a SID register like that is no longer an atomic operation. If SID read accesses are performed from the main program, and SID write accesses are performed in an interrupt routine, problems can arise (e.g. *#2* occassionally reads the result of a SID write operation performed by a music player which was called from the interrupt routine). To avoid that, either disable interrupts before *#1* (can be re-enabled after performing *#2*), or verify (by reading the address of the last accessed SID register in $FD88) that the result of the appropriate access was read from $FD59 (and redo the read accordingly). The second method can be used if interrupts must not be disabled.

```
read:   LDA   $FD59     ← Read SID PotX register (and throw result) (#1)
        LDA   $FD88     ← Read result (#2)
        LDX   $FD89     ← Read number of accessed SID register (#3)
        CPX   #$99      ← Was it ”read register $19”? (#4)
        BNE   read      ← If not, try again (#5)
```

A little bit of warning: these registers always reflect the last SID accesses, regardless to option register settings (ie. SID paddle register remapping in "digital" mouse mode). Reading from $FD59/$FD5A in "digital" mouse mode, result is provided by the card's hardware (digital mouse position counter), but SID register access is still performed, which is reflected by $FD88/$FD89 (likely a $FF, as SID PotX/PotY is disconnected in "digital" mode).

$FD8A,
$FD8B,
$FD8C:
Unused (reads 0)

$FD8D:
Card configuration bits (modify, write-only, reads 0)

Modifying the card's config bits directly, is not possible (for a good reason). Config bits are modified via writing special values to $FD8D.

$D0-$D3: C64 "compatibility" features. Bit 0 switches 886kHz/985kHz mode, bit 1 enables SID access under the $D400 – $D41F range. The combinations are as follows:

$D0: SID is clocked at 886kHz, write access at $D400 – $D41F is disabled (default)
$D1: SID is clocked at 985kHz, write access at $D400 – $D41F is disabled
$D2: SID is clocked at 886kHz, write access at $D400 – $D41F is enabled
$D3: SID is clocked at 985kHz, write access at $D400 – $D41F is enabled

The card defaults to mode $D0, unless the SID Reset button is held while the Reset button of the plus/4 is released; that would initialize the card in mode $D3.

$F0/$F1: $FE80 – $FE9F mirror of the SID register map (Csory SID card compatibility).

$F0: SID at $FE80 – $FE9F is disabled.
$F1: SID at $FE80 – $FE9F is enabled (default).

$DD/$DE: DigiBlaster 8-bit D/A emulation

$DD: DigiBlaster D/A is disabled.
$DE: DigiBlaster D/A is enabled (default).

$A0-$A3: "Mouse mode" and "Mouse type" setting (configure mouse operation). Mouse mode is either "analog" or "digital" ("analog": 1351 mouse and compatibles, "digital": Amiga / Atari ST mouse), mouse type is Amiga or Atari ST. Bit 0 of the config word sets mode, Bit 1 sets type. The possible combinations are as follows:

$A0: Mouse mode: "analog", mouse type: Amiga. This is the default combination.
$A1: Mouse mode: "digital", mouse type: Amiga.
$A2: Mouse mode: "analog", mouse type: Atari ST.
$A3: Mouse mode: "digital", mouse type: Atari ST.

In "analog" mouse mode, the joystick port PotX / PotY inputs are accessible and can be used to interface a 1351 mouse or paddles. With that, the right and middle buttons (RMB and MMB) of digital mice are inaccessible (because their signals and PotX / PotY share the same pins on the joystick port). However, their position can still be tracked because the position counters ($FD82 and $FD83) actually work in this operating mode, and the LMB can also be queried by scanning the joystick fire button.

In "digital" mouse mode, the joyport PotX / PotY pins become RMB and MMB, respectively. In this mode, the SID PotX / PotY registers will be replaced by the digital mouse position counter registers.

The card is capable of handling both Amiga and Atari ST mice, provided that the correct mouse type has been set. The setting is effective even in analog mouse mode (position is updated in $FD82 / $FD83; RMB and MMB are unavailable).

Digital mouse type only works from V1.4 and above. The operation is backwards compatible, that is, on older versions, $A0-$A3 are still interpreted with Bit 1 (mouse type setting) just ignored. On V1.3 and below, the the corresponding bit of the status register (B5) always returns 0.

$E0/$E1: Synergy SID-card compatibility switch

$E0: NST-mode, all registers are accessible (default)
$E1: Synergy SID-card compatibility mode. In this mode the joystick port register is mirrored 16 times in the $FD80 – $FD8F range. None of the NST-specific registers are accessible in this mode. There's a single exception to the rule: the control register can be written $E0, in order to get the card back to native (NST) mode.
Note: Due to a bug, prior to firmware V1.2, bits from registers $FD82/$FD83 and $FD88/$FD89 may get visible in compatibility mode (precisely speaking, these addresses reflect the respective register values OR $FD80, instead of the unmodified mirrors of $FD80). In practice, this shouldn't be a serious problem; it can't even be detected unless there's some device connected to the card's joystick port.

IMPORTANT! Writing the configuration register may be potentially dangerous, because of the following reason:

The joystick port of Solder's SID card uses a 74LS245 8-bit bidirectional bus driver chip. The original plan appears to have been using a MOS 6529B SPI (Single Port Interface) chip here, yet all production cards comprise a 74LS245 (presumably for component availability or component cost reasons). (Incorporating a 6529B would have provided 100% compatibility to the C64 joyport, even including the capability of using the joystick port as output, as supported by the original C64 design.) However, there's a significant difference between the two chips: whilst the 6529B is an NMOS device that only provides weak "high" logical levels (few milliamps max.), the 74LS245 has totem pole outputs capable of sourcing some ten milliamps. In the special case of writing the card's joystick register ($FD80 – $FD8F), the written value is reflected on the joystick port pins for the cycle of the write operation. (The 6529B would also latch and reflect the written value until the next register write or poweroff). There won't be any problems with that, unless the written value contains "1" bits, and the respective joystick port lines are currently being held active ("0") by the external device connected to the port. In that special case, the output of the 74LS245 would source relatively high current. Usually, even that shouldn't be problematic, because microswitches are very unlikely to fail from some ten milliamps of current (and there shouldn't be any risks for the 74LS245 either), yet, active outputs of devices - joystick autofire circuits, 1350/1351 mouse circuitry etc. - could be affected. (Note that driving the output portbits to 0 must be legal in any case, because the original joystick port of the C64 is also capable of sinking relatively high currents, consequently, C64 compatible devices must be aware of that.)

All in all, **it should be verified, PRIOR to writing the configuration register, if the NST SID card is connected to the computer. If the connected card is a Synergy SID card, the program shouldn't attempt writing the joystick port ie. the $FD80 – $FD8F area.** There's an example of the verification method at the end of the register description chapter.

If the card is detected to be a Synergy SID card, that could also be because there's actually an NST card, configured to be in compatibility mode. In that case, one might attempt to switch the card back to native mode, by writing $E0 to $FD8D. Writing $E0 to the joystick port register of the Synergy SID card should be harmless, as bits 0-4 of $E0 are all 0, and bits 7-5 aren't connected to the joystick port of the Synergy SID card, so they couldn't result in any problems.

$FD8E:
Card configuration register (read only, reads $0C by default)

The current configuration setting of the card (read only, can be modified by writing $FD8D).

B7: Computer is in PAL (0) or NTSC (1) mode. The same as bit 6 of $FF07. Test only, don't use.
B6: unused, read 0
B5: "Mouse type" flag: 0: "Amiga", 1: "Atari ST" (V1.4, V1.5+ versions only, others: 0)
B4: "Mouse mode" flag: 0: "analog", 1: "digital"
B3: DigiBlaster D/A flag: 0: disabled, 1: enabled
B2: $FE80 – $FE9F access of SID: 0: disabled, 1: enabled
B1: $D400 – $D41F write access of SID: 0: disabled, 1: enabled
B0: SID clock: 0: 886kHz, 1: 985kHz

$FD8F:
Card "version number" (read only, currently $10, $12, $13, $14, $15)

This register can be used to get version information of the card:

B7: PAL (0) or NTSC (1) type. (The PAL card is unable to generate the correct PAL C64 clock ie. 985kHz with an NTSC plus**/4**, this requires an NTSC card)
B6-B4: Version number of card
B3-B0: Revision number of card
B0: Type of SID (**0**: 858**0**, **1**: 658**1**)

Currently (2014.04.20) the value read from this register is $10, $12, $13, $14, $15 (there's no NTSC version yet, but that one would read $90, $92, $93, $94, $95), which is interpreted V1.0, V1.2 (8580), V1.3 (6581), V1.4 (8580), V1.5 (6581).

This register also makes it possible to detect whether Synergy's or the NST card is present, in a pretty simple manner. The register should be read, and bits B4-B0 masked. If the result is $E0, it's a Synergy SID card. In that case, $FD80 – $FD8F shouldn't be written to (...except for the $E0 value which can't make harm in any way). (There certainly won't ever be V6.x and V7.x versions from the NTSC card, so this check should work 100%).

An example of how to detect the card:

```
            LDA   #$E0        ← Writing this value should be safe
            STA   $FD8D       ← Set NST mode
            LDA   $FD8F       ← Read version register
            AND   #$E0        ← Mask unneeded bits
            CMP   #$E0        ← Are bits B7-B5 all 1? (On the Synergy SID card, they are).
            BEQ   synergy     ← Synergy's card, configuration shouldn't be attempted.
            LDA   #$...
            STA   $DF8D       ← Card configuration...
            ...
            ...
synergy:    LDA #$...         ← Here continues your code
```


Translated by Levente Hársfalvi

2011.09.10.	First version
2012.07.10.	Improve formatting, V1.2, V1.3 FW cards, 6581 option, FW 1.0 bugs
2014.05.10.	V1.4, V1.5 FW cards, Atari ST mouse
2014.06.07.	Mouse *mode* / *type* completion