**MIDIbox SID FAQ**



FAQ is outdated, many details are not matching anymore with the current MIDIbox SID V2 implementation

# Do I need the big and expensive control surface to get some sound out of my SID?

No, you can control all sound parameters of your MIDIbox SID from a computer, with some ergonomic limitations also from a common MIDI controller. However, the dedicated control surface is the optimum solution to control the SID.

# Which PIC is required?

MIDIbox SID 2.0 requires a PIC18F4685.

Previous (1.x) versions required a PIC18F452 for the master and control surface, while the “slave” modules could (optionally) work with the much older PIC16F877.

# Where can I purchase a SID (6581 or 8580)?

Unfortunately the SID has been out of production for years, but you can pull one from any Commodore 64 or Commodore 128. If you don't already own a C64 homecomputer, the "Fleamarket" forum is a good place to start looking.

Other places you might find Commodore computers or SID chips:

- http://www.64andmore.com/
- http://www.jschoenfeld.com/products/cwmk3_e.html
- eBay

# How can I realize a stereo SID synth?

Since SID is a mono device, you can achieve stereo only with more than one SID module - the MIDIbox SID V2 firmware allows to control two SIDs per core, up to 4 cores can be combined to a network for up to 8 SID audio channels.

# What is the purpose of the Audio Input?

It allows you to chain SID modules. Note that the signal from the Audio Input goes also through the multistate filter!
It's also possible to use this input for feedbacking the output. So long the input is not routed through

the filter (FIL→EXT flag cleared) this only results into some distortion. But with the FIL→EXT flag enabled, you will experience a more intensive resonance, which is for example useful for bassline sounds. Details are explained here

## How can I save my sound creations?

If the control surface is enabled, you can save the patch into a BankStick from the SAV menu.

You can also request a dump of a patch with

```
F0 00 00 7E 46 <device-number> 01 <patch-number> F7
```

the returned data can be sent to the MIDIbox SID again in order to reconstruct the sound. The JSynthLib editor comes with an librarian which simplifies the data exchange and which allows you to manage the patches which has been stored on a BankStick.

Update: TL has implemented a really useful “Patch Manager” for MIDIbox SID: http://www.automatic-brain.de/midibox/

## How many patches can be saved in a BankStick?

Up to 127 patches can be saved in a BankStick. They can be selected with the “Program Change” command like known from common synthesizers, or with the datawheel in the mainpage if you've built the control surface.

Up to 8 BankSticks can be connected to MIDIbox SID - makes 1016 patches in total. Banks can be selected with CC #0

Patch #1 of a bank is always the internal patch, regardless if a BankStick is connected or not.

## Do I need a BankStick to get the MIDIbox SID running?

The BankStick is an optional feature. If no stick is connected, only one patch can be stored into the internal EEPROM.

All 128 patch positions are directed to the internal patch.

This is still usefull if you manage the sound patches from the JSynthLib librarian and don't need quick access to your favourite sounds.

## How can I store patches into the slaves

Due to the “master-slave” concept, you need to store the patch into the BankStick of the master (the same PIC which handles the control surface). Thereafter you can transfer it to the slave by sending a

program change command (over the MIDI channel of the slave), or with the control surface: by pressing the appr. SID button and turning the data wheel in the main menu.

## How does MIDIbox SID cooperate with other MIDIboxes?

A MIDIbox can send _F0 00 00 7E 46 <device-number> 0E F7_ which returns all CC parameters. On this way it is possible to refresh the snap values MIDIbox64, will adjust the motorfaders of MIDIboxMF and control the LED rings of MIDIbox64E - from my point of view the ideal interaction between a MIDI controller box and a synthesizer! 🙂

## How can I change the MIDI channel?

If the “Control Surface” is enabled, the MIDI channels of master SID and slave SIDs can be changed within the CFG menu.

If the control surface is not enabled (minimal version of MIDIbox SID), the channel can be changed with following SysEx command:

```
F0 00 00 7E 46 <device-number> 0D 02 00 <channel> F7
<device-number>: from 00 to 0F
<midi-channel>: from 00 to 0F
```

## How can I change the Device ID?

So long the AUTO_DEVICE_ID switch is enabled (this is the default setting), the device ID will be derived from the MIOS Device ID. This is the most comfortable (and less confusing) solution. In this case, the ID cannot be changed from a menu, and also not via SysEx.

If the AUTO_DEVICE_ID switch is disabled, you can change the ID within the CFG menu, or via SysEx:

```
F0 00 00 7E 46 <old device-number> 0D 03 00 <new device-number> F7
```

## How can I create Wavetables?

Wavetables can be edited with the JSynthLib Editor or from the Control Surface. First you should check the example patches which can be downloaded below. The preset library includes some demonstrations of this powerfull feature (patch names begin with WT). You can define up to 32 table entries for 3 different CC parameters which are played with every note. Most common parameters are “Waveform” and “Transpose”, since they allow to define sequences of waves and pitches for percussive sounds.

Four different directives can be selected for each entry: “-” (do nothing), “Play” (change the three parameters), “Goto” (jump to the sequencer-step which is defined as first parameter) and “End” (stop

here).

The three parameter columns are used to enter absolute or relative values for the three CC parameters. Normaly a wavetable begins with absolute values for a defined start when a new note is played (examples: 40 - don't transpose, 3F: minus one note, 41: plus one note. Values can also be entered in decimal form, but a real C64 hacker proscribes such a unusual number system, so both variants are available . For the other values below the first row it's your choice. Pitches (Transpose values) are normaly entered as relative values (begin with a + or -), Waveforms as absolute values (examples: 01 - Triangle, 02 - Saw, 04 - Pulse, 08 - Noise, 12 - oscillator saw & off, 21 - synced triangle). Refer to the SID CC implementation chart.

Here an example for a snare drum:

```
CC Parameter #1 assigned to Voice1 Waveform
CC Parameter #2 assigned to Voice1 Transpose
CC Parameter #3 not used here
Rate (Speed) set to 119

00   Play  08 50 +0   # Plays Noise , transposed by 10 (40 is the middle,
+10)
01   Play  04 45 +0   # Plays a pulse, transposed by 5
02   Play  04 4C +0   # Plays a pulse, transposed by C (=one octave)
03   Play  08 54 +0   # Plays noise, transposed by 54
04   Play  08 +0 +0   # still plays noise, don't transpose
05   Play  18 +0 +0   # stop playing noise
06   End              # stop waveform
```

The result can be heared at the beginning of mbsid_demo8.mp3.

Another example, a synth sound with loop point:

```
CC Parameter #1 assigned to Voice1/2/3 Transpose
CC Parameter #2 not used here
CC Parameter #3 not used here
Rate (Speed) set to 88

00   Play  40 +0 +0   # Setup transpose value (40 is the middle)
01   Play  -1 +0 +0   # decrement transpose value
02   Play  +1 +0 +0   # increment transpose value
03   Play  -1 +0 +0   # decrement transpose value
04   Play  -1 +0 +0   # decrement transpose value once more
05   Goto  01         # define a loop (continue at step 01)
```

This result into a nice dropping sound, known from C64 games :)

However, this was just a short introduction, the possibilities with the Wavetable feature are endless! Some hints: sequence the Filter Cutoff frequency (CC #46), or the Depth of a LFO (CC #72-77) or the Wavetable speed/CC assign value itself (CC#120-124)!

*Update:* since v1.5 it is also possible to trigger notes and arpeggiator keys from the wavetable

sequencer in order to realize more complex textures. The appr. CCs are #8-#11. Examples can be found in the preset patch library (patch #64 upwards).

## How do I start the JSynthLib editor?

This is described in the JSynthLib guide http://www.ucapps.de/jsynthlib.html

## How to supply power?

The recommended power supply for MIDIbox SID is a Commodore 64 Power Supply.

Without the C64 PSU, it's better to use separate transformers: one for the SID module (15V/100 mA should be ok), another for the digital modules (7V/500 mA should be sufficient).

It may work with a single PSU, but the voltage needs to be high enough to supply the regulator on your SID board. Because of the higher input voltage, the 7805 regulators on your CORE module(s) will get very hot, especially with a backlit LCD.

## Will feature X be implemented?

There are several things to consider before a new feature is implemented. Will it be used by a lot of people? Is the hardware really able to do that, will it use up too much memory? Remember, we're only using a PIC here. Also, the MB SID is built in such a way that each editable feature has to have a MIDI CC number. And since most of those are already used, you'll need a *Really Good Reason™* to use up the remaining ones :) In TK's words: *Sounds strange? Here the reason why I'm always very carefully with new features: check the CC list of MIDIbox SID and you will notice that most CC parameters are allocated. Check the content of the SysEx dump (e.g. in the JSynthLib driver) and you will notice that also there most of the memory is allocated (and I already did a lot of tricks).*

*Oh, now you possibly come with the suggestion that NRPN should be used instead of CC — no chance! One of my focus was always the capability to control all parameters via external sequencers/MIDI controllers (most of them cannot handle correctly with NRPN), and via Velocity/Aftertouch/Modulation Wheel/Wavetable. The assigned parameter numbers are stored in 7-bit registers, this limits the maximum number of parameters to 128*

*Next suggestion from your side could be to remove some uninteresting parameters from the CC list to make some new place free for new ones. But this is also nearly impossible, since the internal parameter handling relies on CC parameters. A different implementation would consume much more memory (which means: some nice new functions would possibly never fit into memory anymore)*

Also, using another MIDI channel to make more CC numbers available (again, TK) *won't help, because all parameters have to be stored in the SysEx dump structure, and this is also limited.*

# Can you connect multiple SIDs in parallel to one Core, to get a "unison" kind of sound?

This feature will is realized in MIDIbox SID 2.0, currently in Release Candidate stage.

At present, you can theoretically connect two SIDs to one Core, but since the SID's oscillators are DCOs, they'll pretty much sound the same, which (in theory) will just double the volume. Still, nothing's said until you experiment! Try different configurations, e.g. just putting the SIDs in parallel, or making output of one go to the audio input of the next one to chain filters. Share your findings here on the Wiki!

As TK says: *you can connect multiple SID modules to one core module, in this case they always listen to the same control data. But the audible effect isn't really so interesting. If you are having two identical SIDs, then the volume will just be doubled (the oscillators are digital and therefore always output exactly the same waveform). This option can make sense if you want to switch between 6581 and 8580 because of the different filter characteristics. But for really interesting (especially fat) sounds, seperate cores are the best option, so that the SID parameters are modulated independent from each other. The result is much more analog-like(especially when the LFOs are in freerunning mode and the finetune/portamento/ENV parameters are slightly different).*

# Can we get better envelopes? ATDTLDTSRTR ;)?

This would use up more MIDI CC numbers, and memory. And you know you should take care with that!

# Can we get faster LFOs, WTs?

The SID chip can only be updated at 1220 Hz. Because of that, an LFO starts to “deteriorate” as you make the frequency higher, since there are less and less sample points for each period. If you do want to do stuff at higher frequencies, just use the WT. It goes up to the whole 1220 Hz.

# Is Oscillator FM doable with the SID chips?

As TK puts it: *no*

# Is it possible to add morphing between the patches of MB SID?

It would require memory for a mirror of all parameters (old ones you're morphing from and new ones to morph to). And even more memory for the morphing itself. Conclusion: can't be done really.

## Can you implement modulation depth values in the modulation matrix?

In words of TK: *typical question from somebody who thinks that if other synths have a depth pot for each modulation target, MBSID must have the same. Some technical background: I started MIDIbox SID ca. 3 years ago on a PIC16F877 which doesn't provide a hardware multiplier. Software multiplication takes so much cycles that I had to search for an adequate solution. And the solution was to spend much more LFOs/EGs than common synths and to forward the output values directly to the targets. The result is the same. Not exactly, because common synths mostly don't provide 6 LFOs and two envelopes for a mono voice, they provide seperate depths instead just as a workaround for this limitation. This means also, that there is more live in the sound of MBSID when you just use the existing the modulation possibilities*

## Can you add more sources in the mod matrix (aftertouch, velocity, etc)? Can you add AIN as a mod source?

modmatrix costs a lot of execution time, therefore Aftertouch/Velocity/Modwheel are only assignable to a single target. Using analog inputs is an interesting idea, but I'm not sure if this feature is important enough to disable other functions (it would, again, require another CC).

## Can you add filter resonance as a mod matrix destination?

Have you ever tried sweeping the SID's resonance? :) It's just a two-pole filter. You'll be much more successful with an external filter module.

## Can we have more SIDs in one MB SID?

TK: *As always I have to define such specs before beginning with the implementation. First I thought that controlling two SIDs from one control surface is sufficient for stereo effects. Then I was possibly in such a volatile temper that I decided the incredible: controlling 4 SIDs from a single PIC, which is doing the sound engine of one SID in parallel! Hard to believe that this works without affecting the realtime capabilities too much. I'm really proud that it works without trouble.*

However, if you really want 8 SIDs, TK suggests: *you could build two 4*SID systems... this will possibly cost you 50 EUR more, but what are 50 EUR compared to my unpayable sparetime* 😉

I'd go with that suggestion :)

## How do I make the MB SID sound more phat?

Try external effects, filters. Connect them to the synth's AOUT. Experiment with the Moog filter from Rene Schmitz: http://www.uni-bonn.de/~uzs159/. Try other filters, there are lots of schematics, just type in *vcf schematics* on Google and browse around. Some fine examples are the MS-20 filter and the Oberheim filter. Try guitar effects. It's unbelievable what evil a simple fuzz can do to a lead or a nice “broken” chord. Those things are easy to build, most of the stuff Jimi Hendrix played was three transistors ;). Try ring modulators, try phasers. Those two will always suck in digital, get them in analog or make them in analog :) Besides, doesn't the back of the C64C look like it's just *made* for a patchbay?

You can also beef up the sound of SID's filters by routing the SID output back to its input.

## Does MIDIbox SID support graphical LCDs?

Yes and *no*

MIOS itself supports graphical LCDs, which means that you could use such displays with the MIDIbox SID application.

But the application itself doesn't provide any graphical features - no special icons, no special fonts. So: no benefit at all.

Another problem is, that graphical LCD loads the CPU much more than dotmatrix LCDs, therefore such an option has never been taken into account, it would affect the realtime behaviour of the synth too much.

In theory it should be possible to use a second core just only for the graphical user interface. But no volunteer has been found so far who wants to implement this.

## Do I need an AIN module for the "analog LFO waveform" option?

No, an AIN module is not required.

The analog signal sources can be directly connected to J5:A0..J5:A5 of the core module (unmuxed mode).

If you want to check the analog inputs, just select the “AIN” waveform on one or more LFOs, forward the LFO outputs to the OSC pitch, set the LFO depths to +63, set the LFO rates to 127, and tip your fingers on the analog inputs of the J5 connector. This allows you to modulate the oscillator frequency with your fingers :)

## Why does the "play note" feature of CS Step C play multiple SIDs at once

TK wrote: Thats a known “problem”. I had to decide if I either send a “play command” via SysEx, or

via a common MIDI note. I prefered to use a MIDI note, since it is transfered faster (minimum latency) and since it is compatible with the old PIC16F hardware

Sending a note instead of SysEx has also the advantage, that if SIDs are assigned to the same channel, they will be started at exactly the same time without a delay. Doing the same with SysEx would mean that I would have to implement something like a “broadcast” function (SysEx message which is taken independent from the Device ID), and this would increase the code size too much for such a minor feature. For the PIC16F firmware it wouldn't be possible (out of memory)

## Why must MIDIbox SID stop playing a note when a patch is changed

Changing this behaviour would lead to many unwanted side effects and wouldn't work properly in all cases. A patch change not only refreshes the SID registers, but essentially resets the whole sound engine. There are many dependencies between the registers (values which have to be stored and written back after the change), which makes such a feature a mess for a programmer.

A proper patch change without reset would require that the whole patch is transfered to the SID as well as to the sound engine within one update cycle (0.8 mS), which is not possible with the PIC. It's especially not possible to send so much data to the slaves within this short timewindow, since the transfer takes about 260 ms.

## How is it possible to save CC changes which have been sent from external

Short answer: enable the edit mode. In this mode, all CC changes which have been sent to the master will be regognized by the control surface. CC changes to the slaves won't be regognized.

Long answer - TK wrote: Update of CCs to slaves: nearly impossible with the concept. At the time where I started with the implementation of the SID application on the PIC18F I had to decide how to store the data structures. I choosed a dual storage method: one compressed structure which reflects the SysEx dump and which is easy (and fast) to handle for the control surface when multiple SIDs have to be serviced, and another structure which is optimized for the sound engine. A third structure is given by the CC→parameter assignments, but this is (fortunately) not stored in memory, but directly transfered into the SID and SysEx memory.

This works fine in most cases, but it has disadvantages which must be accepted. E.g., since it is possible to change multiple parameters with a single CC (e.g. CC#16 which changes the transpose value of all three oscillators), an incoming CC cannot be easily mapped to a byte in the SysEx structure. The current solution is to read back the data from the sound engine data storage and copy it into the SysEx storage when the CPU has some free time. This works fine, but only for the master SID, because the control surface has no access to the sound engine storage of the slave SIDs (unidirectional interface…)

Another solution could be to write a large function which decodes the incoming CCs and passes them to the appr. registers of the SysEx structure. This would work for the master and the slaves, but it would consume so much code memory, that other features would have to be removed. Therefore I

prefered the “cheaper” solution.

There are two good points which I want to mention here: 1) for the MIDIbox FM I solved such problems very early in the design phase - MBFM has a single data structure which is optimized for SysEx/CC and OPL3 accesses (it took some time to find out all relationships) - so, in MBFM the data is always consistent 2) sometimes I'm thinking about a major redesign of the SID application which doesn't consider compatibility issues, but results into an “perfect” implementation based on my experiences in the last years. But on the other hand I think that the current implementation isn't that bad, that everybody can live with such imperfections when he knows how to handle with it...

However, just another remark: if the PIC18F4620 wouldn't have that f*cked EUSART and CPU bugs, all these issues wouldn't exist anymore, because this chip offers twice the flash memory, and much more RAM... the existing firmware could be freely extended without such “50% solutions”. So, my hope is, that Microchip releases a new chip revision, so that it is possible to make the application perfect without the previously mentioned redesign.

## What parts can be left out from the core modules when building a multi SID synth?

Thanks wilba for the tip, you can remove everything to the left of the regulator including the regulator shown on the multi sid PDF. Remember to bridge the core PCB's where the regulators used to be. I have also heard of swapping the 2200uf cap, and not including the bridge on the cores too but I have not tested it (dcer10). Also remember when you are testing the multi sid that you need to press the link button to hear the extra sids!

From:
https://www.midibox.org/dokuwiki/ - **MIDIbox**

Permanent link:
**https://www.midibox.org/dokuwiki/doku.php?id=midibox_sid_faq&rev=1197062661**

Last update: **2007/12/07 21:50**