# CBUS-enabled Sound Player

## Construction & Usage Notes – for PCB revisions D-F and software version 2.x

### Overview

The Sound Player is a layout module that plays MP3 audio files under MERG CBUS control. It can learn CBUS events containing lists of up to sixteen audio tracks or commands, and play them back in sequences of up to 64 tracks. Up to 128 events can be learned. It operates in both SLiM and FLiM modes.

### Construction

These notes assume some familiarity with electronics design and construction, the Arduino system, and MERG CBUS.

**NB There is an error on board revisions D-F.** Pin 7 of IC6 (the 24LCxxx external EEPROM) should be connected to 0V. This can be achieved by soldering a wire on the reverse of the board from pin 7 to any of pins 1, 2, 3 or 4, during the final stages of construction. Cover it with some electrical tape to protect the wire.

There are a number of options that you should consider before purchasing components and starting construction.

A. The board requires a 12V DC power supply. The 5V voltage regulator circuit is limited to 1A. The voltage regulator IC1 will almost certainly require a heatsink.

B. Component selection:

- A bill of materials (BOM) has been provided with recommended sources for components.
- Resistors R1, R2 and R3 provide current limiting for the three LEDs. Choose values based on the current rating of the LEDs you have and the brightness you want. The circuit shows 1K but bright LEDs may need 2K2 or higher. You should experiment or use ledcalc.com
- Resistors R9 and R10 (if shown) are not required and should not be fitted.
- Capacitor C7 (if shown) is not required and should not be fitted.
- R12 is a 120-ohm CAN bus termination resistor, which can be fitted if desired. The associated jumper can be closed to enable the termination, either permanently by soldering a link or temporarily by shunting.
- Polarized capacitors C1 and C9 are not critical and can be whatever electrolytics or tantalums you have to hand. Choose something like 47uF to 100uF for C1 and around 10uF for C9, with appropriate voltage ratings.

- The remaining capacitors can be of any ceramic type.

- Switch SW1 is not necessary if you are using a rotary controller with a built-in push switch; they perform the same function.

C. You can mount the 'user interface' components off-board, e.g. on a control panel or box lid. Just run hookup wires to the appropriate locations on the PCB.

D. The module can be operated 'headless', i.e. without the rotary encoder and the display, if you plan to configure it from FCU. You can omit the two 7-segment displays, the rotary controller, IC4 (MAX7219), R5, R6 and R13, C7, C8, C9, C10 and C11. You will need to fit switch SW1 and the three LEDs. The module can then be hidden away as desired. The software does not need to be amended.

E. The sound-to-light functionality is optional, comprising transistor Q1 and resistor R14, as well as the PCB terminals. Q1 can be any suitable logic-level N-channel MOSFET rated for the load you intend to switch, e.g. IRL520. These components can be omitted if desired.

F. The transient suppression diode (D4) protects the CAN transceiver IC from any stray high voltages on the CAN bus. It is a tiny surface-mount (SOT-23) device and tricky to solder, but highly recommended if you can fit it.

G. The external serial EEPROM (IC6) is now mandatory as of version 2 of the software. Various EEPROM sizes are available in pin-compatible packages. My test build uses a 24LC128, although the software is limited to 128 learned events, which consume just over 5K of memory. Fitting a 512Kbit part will enable bootloading over CBUS in an upcoming version of the software.

   Minimum size is 64Kbits (8Kbytes). 128Kbits = 16Kbytes, 512Kbits = 64Kbytes.

H. Footprints are provided on the PCB for MERG standard 3.5mm pitch screw terminals, but you can use whatever connectors you prefer, or solder wires directly to the PCB.

I. A 6-pin 0.1" header is provided for uploading new software and accessing the serial console. You will require a USB-Serial adapter, available on eBay for a couple of pounds, e.g. https://www.ebay.co.uk/itm/USB-to-TTL-UART-CH340-Serial-Converter-Module-3-3-5V-with-DTR-pin-lead-UK-FAST/181672265616

J. The SPI and I2C headers are not currently used, but are provided for the convenience of possible future enhancements.

K. 5V and 12V are available as outputs; take care not to overload the power supply.

## Basic Setup

You will need a micro SD memory card formatted as FAT16 or FAT32. It seems impossible to buy anything smaller than 8GB these days, even though you won't use 10% of that. There are no special performance requirements: we're not shooting 4K video, so you could recycle one from an old smartphone or camera.

You will also need to be able to insert the card into your PC to transfer files. If it doesn't have a dedicated slot for memory cards, you can get a USB adaptor from the pound shop or eBay.

I highly recommend downloading and using the official formatting program from the SD organisation's website: https://www.sdcard.org/downloads/formatter_4/index.html

Create a directory/folder on the blank card named */MP3* and copy over your MP3 files in this. Name the files after the track number you will refer to them by, starting from zero, e.g. *0000.MP3*, *0001.MP3*, *0002.MP3*, ..., *0133.MP3*, etc. The maximum supported number is 237, or *0237.MP3*

It would be well worth keeping a spreadsheet or other record of what sounds each file contains.

Insert the card into the MP3 player module, select a track number (00 - 99) using the rotary encoder and push the switch to play. Pressing the switch again whilst a track is playing will stop playback.

At start-up, the display briefly displays the node number and then defaults to track 0. The display shows 00 if a card is present and recognised, or nc if a card is not present or is unrecognised.

## Audio Output

There are two audio output options:

a. the MP3 player module has an on-board 3W mono amplifier which will directly drive an 8-ohm speaker from outputs SPK 1 and SPK 2. Extended use of a lower impedance speaker may overload the voltage regulator unless you place a suitable resistor in series with one of the speaker outputs

b. pre-amp (line) level stereo output is available at DAC L and DAC R. These can be connected to an audio amplifier. Use decent quality screened cable and experiment with grounding the screen at either or both ends

You can find cheap amplifier modules on eBay for a few pounds. Or use an existing amp or powered speakers.

## Sequences and Chains

Two fundamental concepts that are worth getting to grips with are *sequences* and *chains*.

A '*play sequence*' is the list of up to 64 tracks and/or commands that are currently being played.

Each learned CBUS event can have up to 16 associated numbers (*Event Variables* or *EVs*). Number values can range from 0 to 255 (the maximum range of an 8-bit byte).

Each of these 16 numbers represents either an audio track (0-239) to be played, or a command (240-255) to be executed. On receipt of a previously-learned event, the associated numbers are added to the current play sequence as described below, and the module begins playing.

If a command (240-254) is the *first and only* EV associated with an event, it is executed immediately. This may be useful for creating a physical control panel for the module as you could then configure pushbutton switches for Pause, Skip, Mute, Stop, Volume, etc.

If a command appears anywhere else, it is added to the play sequence just like any other track, and is executed in sequence. This is useful for e.g. setting the volume level or EQ before a specific audio track, or switching the sound-to-light output on and off between tracks.

If you need sequences longer than 16 tracks (for example, if you are joining fragments of speech together to form sentences) this can be achieved in two ways.

### *Using Node Variables*

The first approach uses a module *Node Variable* (*NV*). CBUS module node variables are designed to be power-on defaults and are programmed using FCU when in FLiM mode. In this module, they apply unless overridden by a command.

Node variable NV10 controls event chaining and can be set to either *true* (1) or *false* (0). If set to true, the tracks associated with any subsequent events will be added ('*chained*') to the end of the current play sequence, thus extending it.

For example, a learned event contains the sequence 5, 6, 7, 255. On receipt of this event, the module will add the three tracks 5, 6, 7 to the play sequence and begin to play them out (255 indicates 'end of list'). Whilst this sequence is playing, you can send another event containing another list of tracks.

If NV10 is true (1), the tracks are added at the end of the play sequence. So, if you were to send the same event again, the play sequence would then become 5, 6, 7, 5, 6, 7. This is the default behaviour unless changed.

If NV10 is false (0), the current sequence is overwritten by the next event. For example, we send an event (5, 6, 7, 255) and the play sequence contains tracks 5, 6, 7. Whilst track 5 is playing, another event (10, 20, 30, 255) is received. Instead of adding these tracks to the end of the play sequence, they overwrite the unplayed tracks (6, 7) and the play sequence becomes 5, 10, 20, 30.

### *Using a Command*

The second approach to creating playing sequences longer than 16 tracks uses a command EV inserted in an event's track list (command 254). The event variable following the number 254 is interpreted as an event index to be chained, and the tracks from that event are added after those of the current event. For example:

*Event 3 contains the sequence: 5, 6, 7, 8, 9, 10, 254, 4, 255*
*Event 4 contains the sequence: 10, 20, 30, 40, 255*

This causes tracks 5 to 10 from Event 3 to be added to the sequence, followed by the tracks 10, 20, 30 and 40 from Event 4, for an overall, combined sequence of ten tracks.

In either case, the maximum sequence length is 64 tracks, so you are protected from infinitely-recursive self-referencing chains!

### *Immediate Play Mode*

The value of NV13 determines whether the currently playing track is interrupted after a change of sequence ('immediate play').

If set to false (0), the current track keeps playing until the end and then continues with the amended sequence. This is the module default, unless changed.

If NV13 is set to true (1), the currently playing track is immediately stopped and play continues with the next track in the amended sequence. This is useful when you want to interrupt playback for an urgent sequence.

This behaviour can also be controlled from a learned event using EV command 251.

### *Loop Mode*

Command 249 sets the module into loop mode. The current play sequence is repeated (looped) *ad infinitum*, or until another event is received.

## Initial Configuration

As we cannot predict the EEPROM contents of any IC we may use, a method has been provided for initialising the storage and setting a unique node number. This should be done as the first step and before the module is attached to your layout's CBUS.

a. hold the push switch whilst powering on the module
b. the display will blank, and both green and yellow LEDs will light
c. release the switch and then press for a further 5 seconds (avoids accidents!)
d. the display will change to ⌐⌐
e. select the desired Node Number (0 1-99) using the rotary encoder
f. press the switch again for a further 2 seconds to initialise the EEPROM(s)
g. the display shows – – whilst this process takes place (up to 30 seconds)
h. the module now continues the start-up

By default, a SLiM module's CAN ID is set to the same as the Node Number, so ensure that this number is unique on your layout.

There is an inactivity timeout which will abort the reset/setup process if you don't complete the operation within 30 seconds. Or you can simply power off the module at any time.

The module's node number is briefly shown blinking on the LED display each time the module starts up.

**NB** the setup process will delete any stored events, so make sure you have a backup in FCU if this is important to you.

**NB** the reset method only works if the module is in SLiM mode. If it is in FLiM mode, return it to SLiM mode first before resetting, to save confusing FCU.

## SLiM Mode

To learn an event in SLiM mode:

- ✓ press the switch for two seconds and release
- ✓ the display changes to LL
- ✓ use the rotary encoder to choose a track number from 0 to 99
- ✓ send an ACON or ACOF event from a producer module
- ✓ the event will be learned and the selected track number stored in sequence
- ✓ you can repeat this process a. to e. to learn the same event multiple times and build up a play sequence of up to 16 tracks
- ✓ to test, send the event again and the track or sequence should be played

To unlearn an event in SLiM mode:

- ✓ press the switch for two seconds and release
- ✓ when the display shows LL, press the button once more to change to unlearn mode

- ✓ the display will change to ␣␣
- ✓ the switch can be pressed to cycle between learn LL and unlearn ␣␣ modes
- ✓ send the event to be unlearned
- ✓ the event (and its associated track sequence) will be unlearned
- ✓ to test, send the event again, which should result in no action

NB: events are learned as a unique combination of node number (NN), event number (EN) and opcode (OPC). In this way, an ON event is different from an OFF event and they are learned separately, with separate play sequences. Currently opcodes ACON (0x90) and ACOF (0x91) are supported. This behaviour conflicts with CBUS rules and will upset FCU if the module is later switched to FLiM mode.

## FLiM Mode

The module masquerades to FCU as a CANLED64, a consumer-only module with 64 outputs. This is sufficiently generic to mean FCU does not need to be updated to specifically support this module.

A. Switching to FLiM mode follows standard MERG CBUS practice:

a. press and hold the switch for 6 seconds
b. the yellow LED will blink
c. release the switch
d. use FCU to allocate an unused node number and give the module a descriptive name:

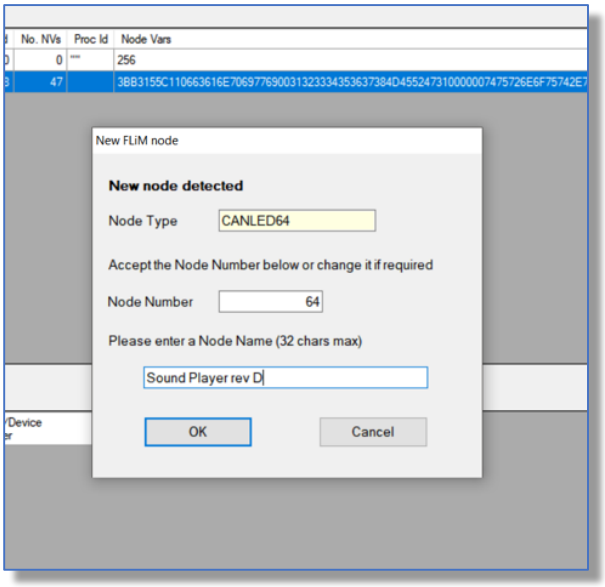


**Fig 1. Allocate a node number and give the module a descriptive name**

e. the green LED goes off and the yellow LED remains on
f. gather the module's configuration using FCU:
   - select the module in the list
   - alt-P to read its node parameters
   - alt-R to read its node variables
   - alt-E to read any previously learned events

B. To teach an event from FCU, first generate one or more events from a producer module.

As the module is not really a CANLED64, we must use FCU's generic dialog capability. From the menu, select ***Settings -> Advanced -> Use Generic Dlgs***. This option is now ticked in the menu.

Select the event to be taught to the module and choose ***Events -> Teach...*** from the menu. Select your module from the list of available modules and click OK. Or drag and drop the event on to the module.

The generic Event Teaching dialog box is now displayed:

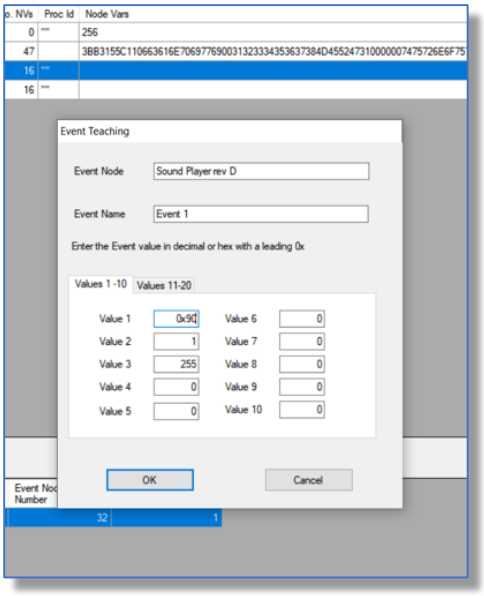


**Fig 2. Teach an event to the module**

The 'Value' boxes should be filled in as follows:

a. Value 1: the event opcode to learn, e.g. 0x90 for an ACON (accessory on) event or 0x91 for ACOF (accessory off)

b. Values 2 - 17: up to 16 track or command numbers to be played in sequence

**NB** if you want to store fewer than 8 (or 16) tracks, be sure to make your final entry 255, meaning 'end of list'. Otherwise FCU will interpret empty boxes as 0, and the module will repeat track 0 several times ... or give errors trying to!

In the picture above, the single track (1) has been associated with opcode 0x90 (ACON). The sequence is terminated by entering the number 255 for Value 3.

**NB** FCU does not permit teaching the same event to a module more than once. If you want separate sequences for the same event's ON and OFF variants, you must program these manually in SLiM mode (or edit the source code to store the desired configuration in EEPROM).

PS if you want to join up fragments of speech, make sure to edit the 'blank space' at the beginning and end of each sound file. The lag between one track finishing and the next one beginning is only a few microseconds, but the gap may seem much greater if your tracks have a long lead-in or lead-out.

## Produced Events

The module can produce events at the beginning and end of each play sequence. This behaviour is controlled by NV11. If set to true (1, the default), the module produces an ACON event at the beginning of each play sequence and ACOF at the end. The event number (EN) is zero (0x00 0x00).

## Node Variables

Sixteen Node Variables (NVs) have been defined for non-volatile settings. They are applied at start-up and can be updated using FCU (defaults in parentheses):

01 – player volume 0-30 (5)
02 – EQ setting - 0-5, 0: normal, 1: pop, 2: rock, 3: jazz, 4: classic, 5: bass (0)
03 – MP3 player comms. timeout in units of 10ms (50 = 500ms)
04 – sound-to-light output, (1, on)
05 – STL high threshold high byte (0x02)
06 – STL high threshold low byte (0x0D)
07 – STL low threshold high byte (0x01)
08 – STL low threshold low byte (0xDB)
09 – LED 7-segment display brightness 0-15 (5)
10 – sequence chain: whether to add the next event to the end of the current sequence or immediately after the currently-playing track (1, true)
11 – whether to send a beginning and end of sequence CBUS event (opcode ACON / ACOF, EN = 0)
~~12 – use external EEPROM for event storage (1)~~
13 – immediate play, i.e. whether to interrupt any current play sequence with the next event (0, false)
14 – track number to play on start-up; 255 = don't play a track
15 – tba
16 – tba

## Commands

Whilst track numbers 0-255 are available, the final 16 have been reserved for commands for controlling the player:

240/F0 – increase volume one step (max 31)
241/F1 – decrease volume one step (min 0)
242/F2 – skip to next track in current play sequence, if any
243/F3 – stop playing current track and clear any learned sequence (emergency stop)
244/F4 – pause or resume playback
245/F5 – volume mute

246/F6 – volume unmute
247/F7 – STL output on
248/F8 – STL output off
249/F9 – loop mode, repeats the current sequence until interrupted
250/FA – set volume level (to the value in the next EV in the learned event) (0-31)
251/FB – immediate play: do not wait for current track to finish playing
252/FC – hold mode: do not play the sequence until released
253/FD – un-hold: release from hold mode
254/FE – chain sequence (the following EV is an event index whose tracks will be added to the play sequence at the end of this list)
255/FF – used as a null or no-op value (instead of zero) to mark the end of a sequence in a learned event

*See the section on Chains and Sequences above for more information.*

## Console

Copious information is available on the serial port. You can use the Arduino IDE's serial monitor or your preferred terminal program. Communications settings are 115200 bps, 8 data bits, no parity, 1 stop bit (115200 8N1). If you have problems connecting, try swapping around the TX and RX cables.

A number of single-character commands are available; just press the key:

Configuration:

- **n** – displays the CBUS node configuration & software version
- **e** – displays the EEPROM configuration and stored events**
- **v** – displays the node variables
- **c** – displays the CAN bus status
- **m** – displays the MP3 player status
- **s** – displays the sound-to-light status

Player commands:

- **u** – volume up
- **d** – volume down
- **q** – cycle through EQ settings
- **k** – skip to the next track in the current play sequence
- **!** – emergency stop all tracks
- **t** – mute and unmute (toggle)
- **p** – pause and resume (toggle)

*** You can copy and paste this data into a text file and import as CSV format into Excel (using the pipe 'I' character as a delimiter).*

## Loading Software Updates

You will need some familiarity with the Arduino IDE to upload software updates from source code. You will also need to download some 3rd party libraries that are used. See Appendix.

Alternatively, it may be possible to provide a binary distribution but a means of uploading will still be needed (other than exchanging pre-programmed ICs).

Under the covers, the Arduino IDE uses an open-source command-line program named ***avrdude*** to perform the actual code upload to the MCU from a USB/serial port. It is available for Windows, Mac and Linux.

Windows users could try this utility: http://russemotto.com/xloader/, which provides a friendlier interface.

### Warning Messages

A number of text messages are shown on the LED display, either static or blinking:

- nc = no card found, or the card is unreadable
- EE alternating with a number = an error from the MP3 player, usually an attempt to play a non-existent sound file

### Summary of Switch Usage

In SLiM mode:

- a one-second press plays or stops the currently displayed track
- a two-second press sets the module in to learn/unlearn mode
    - further short presses alternate between learn and unlearn modes
- a six-second press will attempt to change to FLiM mode
    - to abort this change, press the switch again for two seconds

In FLiM mode:

- a brief press requests a new Node Number from FCU
- a brief double-press starts the CAN ID enumeration process
- a one-second press plays or stops the currently displayed track
- a six-second press releases the node and reverts to SLiM mode

See above for more detailed discussion of these operations.

### Summary of LED indications

Green:

- solid on to indicate the module is in SLiM mode

Yellow:

- blinking to indicate transition from SLiM to FLiM modes
- solid on to indicate the module is in FLiM mode

Both LEDs on/blinking in SLiM setup/reset mode.

## Contact

Comments and feedback are welcome. For assistance, please contact me on the MERG forum where my username is ***sharp5***.

My personal contact details are:

Duncan Greenwood (M5767)
Cambridge, UK

m: +44 7449 603 906
e: duncan_greenwood@hotmail.com

## Licence and Warranty

## Appendix

The following third-party libraries required for source code compilation.

These libraries are used by the project, in addition to the standard Arduino libraries.

**Streaming** - C++ stream style output, v5
http://arduiniana.org/libraries/streaming/

**DFRobotDFPlayerMini** - DFPlayer Mini MP3 player, v1.0.2
https://github.com/DFRobot/DFRobotDFPlayerMini

**LedControl** - MAX7219 / 7-seg LEDs, v1.0.6
https://github.com/wayoda/LedControl

**MCP_CAN** - CAN Bus library using MCP2515, v1.5
https://github.com/coryjfowler/MCP_CAN_lib)

**extEEPROM** - I2C EEPROM library, v3.3.5
https://github.com/JChristensen/extEEPROM