This page is to describe the operation of the MB808 Sequencer application. It is a work in progress, comments are welcome and usually appreciated :)

Forum Thread: http://www.midibox.org/forum/index.php?topic=7391.0

This document is split into 3 sections:

- Application Summary
- Operation Manual
- Technical details

Application Summary

This application is not meant to be part of a modular system or even a standalone box, it's origins are a simple adaptation of the MBSEQ v2.4 application to mimic to functionality of the original 808's digital section. As such the primary task for this application is to provide 13 editable tracks to cover the 12 instruments + accent data and to create the proper pulse signals for the inputs of the instruments. However, MBSEQ v2.4 is a very deep application and it would be a shame to ignore many it's features so the primary task is to create an interface using the available buttons and encoders that can access as many menu functions as possible, without making the whole thing obscure. Remember, this is a drummachine first, sequencer second :)

Features

This is a list of planned features. Certian to be changed over time.

- Standard 16 step editing format. Accent on it's own “track”.
- Variable pattern length from 1-32 steps
- Pattern chains can be created in song mode
- All track parameters available to the SEQ V2.4 application in drum mode will be available. The plan is to support all track parameters of SEQ v3
- Edit mode can be toggled on and off in realtime
- Song mode will be as robust as the interface allows
- Support for LCD's will be kept in software
- 8 banks with 8 patterns each
- Each pattern will have A and B variations that can be switched between or morphed between with dedicated controlls
- Save, Copy, Paste and Clear operations. One level of undo would be sweet.

Theory of operation

The MB808 application will essentially be unchanged from the SEQ v2.4 application. The changes that will be made reguard the interface, specifically the LCD screen and encoders will not be present. At this time I also think it is best to use TK's drummode for the operation of the sequencer. Although this prevents each step from having it's own flam value, what we gain is much more than what we loose. Firstly using drum mode gives us 12 instrument tracks + accent all in one pattern as opposed to re-working the handling of the pattern sets to allow the use of 12 complete tracks, this keeps things

compatible with the base application and hopefully with the upcoming SEQv3 application :) Using drum mode also allows us to use the morph feature, but in a way that mimics the original 808. Each pattern can be linked to the same pattern in the lowercase bank switched between like with the “A+B” toggle switch on the original, my favorite feature. This will take a bit of trickery in code to keep things compatible, so it may not be implemented in the first version. The second big advantage I realized this morning while thinking about the Pro-1. It's sequencer was absolutely minimal, you just switched into edit mode and entered notes until the phrase was entered. No rests. The Pro-1 would then play it back, automatically looping it. I realized that with the extra tracks avaiable and using the arrpegiate mode along with the step edit mode the mb808 could also sequence basslines and trigger arrpegios! This is very cool!

Description of the interface

There are 35 elements to the MB808 interface. They are:

- 16 GP buttons
- 4 Menu buttons (m1 - m4)
- 4 Mode buttons (Song, Pattern, Mute, Edit)
- 5 Transport controls (Play, Stop, Fwd, Rwd, Loop)
- 2 Control buttons (Select, Exit)
- Instrument Encoder
- Tempo encoder
- Swing/Morph pot
- Datawheel

Operation Manual

NOTE: this is still a hypothetical interface. The changes reguarding V3 are not represented here and this is really just to show how the F buttons can be used as menu buttons to reduce dependancy on an LCD screen.

Modes of operation

The MB808 application has 5 different modes that it can operate in, they are:

- Song Play
- Song Edit
- Pattern Play
- Pattern Edit
- Mute

In each mode the 16 GP buttons have a different function and each mode has it's own set of menus accessed with the menu buttons. The operation of the 16 GP's for each mode and menu are described below

Pattern Play																

GP#	GP1	GP2	GP3	GP4	GP5	GP6	GP7	GP8	GP9	GP10	GP11	GP12	GP13	GP14	GP15	GP16
Default	Bank 1	Bank 2	Bank 3	Bank 4	Bank 5	Bank 6	Bank 7	Bank 8	Pattern 1	Pattern 2	Pattern 3	Pattern 4	Pattern 5	Pattern 6	Pattern 7	Pattern 8
F2	Forward	Backward	Ping Pong	Random	BPM / 1	BPM / 2	BPM / 4	BPM / 8	BPM / 16	BPM / 32	BPM / 64					

Pattern Edit																
GP#	GP1	GP2	GP3	GP4	GP5	GP6	GP7	GP8	GP9	GP10	GP11	GP12	GP13	GP14	GP15	GP16
Default	Step 1	Step 2	Step 3	Step 4	Step 5	Step 6	Step 7	Step 8	Step 9	Step 10	Step 11	Step 12	Step 13	Step 14	Step 15	Step 16
F1	BD	SD	LT/LC	MT/MC	HT/HC	CP	MA	RS/CL	CB	CY	OH	CH	Control 1	Control 2	Control 3	Control 4
F2	Forward	Backward	Ping Pong	Random	BPM / 1	BPM / 2	BPM / 4	BPM / 8	BPM / 16	BPM / 32	BPM / 64					

Song Play																
GP#	GP1	GP2	GP3	GP4	GP5	GP6	GP7	GP8	GP9	GP10	GP11	GP12	GP13	GP14	GP15	GP16
Default	Song 1	Song 2	Song 3	Song 4	Song 5	Song 6	Song 7	Song 8	Song 9	Song 10	Song 11	Song 12	Song 13	Song 14	Song 15	Song 16
F1	Position 1	Position 2	Position 3	Position 4	Position 5	Position 6	Position 7	Position 8	Position 9	Position 10	Position 11	Position 12	Position 13	Position 14	Position 15	Position 16

Song Edit																
GP#	GP1	GP2	GP3	GP4	GP5	GP6	GP7	GP8	GP9	GP10	GP11	GP12	GP13	GP14	GP15	GP16
Default	Bank 1	Bank 2	Bank 3	Bank 4	Bank 5	Bank 6	Bank 7	Bank 8	Pattern 1	Pattern 2	Pattern 3	Pattern 4	Pattern 5	Pattern 6	Pattern 7	Pattern 8
F1	Position 1	Position 2	Position 3	Position 4	Position 5	Position 6	Position 7	Position 8	Position 9	Position 10	Position 11	Position 12	Position 13	Position 14	Position 15	Position 16
F2	Part 1	Part 2	Part 3	Part 4	Part 5	Part 6	Part 7	Part 8								

Mute Play																
GP#	GP1	GP2	GP3	GP4	GP5	GP6	GP7	GP8	GP9	GP10	GP11	GP12	GP13	GP14	GP15	GP16
Default	BD	SD	LT/LC	MT/MC	HT/HC	CP	MA	RS/CL	CB	CY	OH	CH	Control 1	Control 2	Control 3	Control 4
F1	Group 1	Group 2	Group 3	Group 4	Group 5	Group 6	Group 7	Group 8	Group 9	Group 10	Group 11	Group 12	Group 13	Group 14	Group 15	Group 16

Global Functions																
GP#	GP1	GP2	GP3	GP4	GP5	GP6	GP7	GP8	GP9	GP10	GP11	GP12	GP13	GP14	GP15	GP16
F3	Bank 1	Bank 2	Bank 3	Bank 4	Bank 5	Bank 6	Bank 7	Bank 8	Pattern 1	Pattern 2	Pattern 3	Pattern 4	Pattern 5	Pattern 6	Pattern 7	Pattern 8
F4	Next bar	Next quart.	Next 8th	Next 16th		Clear Patt	Clear Track	Copy Track	Copy Patt	Paste	Save	MIDI	SysEx			

Button	Loop	Fwd	Rwd	Stop	Play	Tap	Ins. Select	Datawheel
Pattern Play	Set loop points	Next MetaBank	Prev MetaBank	Stop	Play	Tap	Ins. Select	
Pattern Edit	Set loop points	Next Bar	Prev Bar	Stop	Play	Audition	Ins Select	Meter
Song Play	latch loop	Next Pos.	Prev Pos	Stop	Play	Tap	Ins. Select	
Song Edit	Jump point	Next MetaBank	Prev MetaBank	Stop	Play	Tap	Ins. Select	

Technical Details

There are 3 main components to this modification:

- The sequencer needs to output 1ms active high pulses to the gate inputs of the 12 instruments and a 1ms active low pulse to the common trig circuitry which creates the accent signal.
- The interface needs to be modified so that the LCD and datawheel are not neccessary

- Special functions need to be added to the application to steamline operation

Connecting with the analog circuits

The 808's instruments are all comprised of a combination of twin-t oscillators, a noise source and simple envelopes. Both the twin-t and the envelope take a breif pulse of varying amplitude to trigger and set the amplitude and each of the 808's instruments have a AND gate made up of 2 transistors to interface with the digital controller. These AND gates have inputs for 2 signals, called common trig and instrument data. Each instrument has it's own data line but there is only one common trig which is shared by all instruments. The common trig signal is active high for 1ms in synchronization with the master clock. Each instrument data signal is a 5v active high 1ms pulse which is only present when that instrument should be triggered. The AND gate insures that both signals are present before sounding an instrument. Creating the instrument data signal is easy using MB hardware, a DOUT pin does the job nicely. Altering the code to flash a DOUT pin on a note event is fairly simple as well. The common trig signal is a little more complicated. The accent data, which is a voltage between 5v and 15v is also represented in the common trig signal. As such it is the pulse present on the common trig input that triggers the instrument while the instrument data signal insures the instrument is only triggered at the rigt step. Luckily the accent signal only needs two states, on and off, where off is always 5v and on is a higher voltage determined by a potentiometer. Now, this could be solved by using a logiv switch like the 4016 or 4066, using a DOUT pin to toggle between connecting the common trig line to 5v or the wiper of the accent pot. However, looking over the schematics I've found an even simpler way that only needs a couple transistors. The concept is simple, we still use a pot connected as a voltage divider between the 5v and 15v rails to get our accent voltage but a simple transistor can be used to shunt the voltage of the wiper to the 5v rail when it is turned on. If we invert the common trig signal in code so that it is active low we only need the single transistor ;)

Now, in the original, the common trig pulsed on every clock tick. This isn't neccessary, instead the common trig line will only be pulsed when a note on occurs.

Modifying the interface

In order to get the most out of the SEQ application without using a datawheel or an LCD and without risking becoming totally incompatible with future versions I decided it was best to use the 4 function buttons as menu buttons to modify the action of the GP's. In order to do this I need to modify the code in each of the “F” buttons handlers. These can be found starting on line 681 of seq_buttons.inc. First I cleared out the code that was in there. Now, the simplest way to do this is to set a bit representing when one of the menu buttons is pressed. Since I want them all to be momentary I can then simply clear the bit when the button is released. Code will be needed to be sure that only one bit is set at a time as it makes no sense to edit multiple menu's at once. Now, in order to turn these button presses into additional menu shortcuts the operation of the GP's need to be overloaded with 4 additional cases represented by the bits in the “current_menu” register. Since the operation of the GP buttons and the menus is not exactly straight forward care needs to be taken so that the normal operation of the SEQ application is not interrupted. SO, how, exactly do the GP menu shortcuts work?

- Once a DIN event has been determined to have come from one of the two SR's assigned to be the GP buttons SEQ_GP_Button [21] is called.
- In SEQ_GP_Button [21] the application can branch to 3 different places.
- If Menu mode is active (the menu button is pressed) than the app branches to SEQ_GP_Mode4_Button [240]

This is done using one of TK's macros: “BIFSET” which checks if a specified bit in a specified register is set and then branches to a specified function if it is. The register is SEQ_MODE0, the bit for Menu mode is SEQ_MODE0_MENU. The line of code: `BIFSET SEQ_MODE0, SEQ_MODE0_MENU, BANKED, rgoto SEQ_GP_Mode4_Button`. The functionality we want to add is very similar to the Menu shortcut mode, but lets not jump there just yet, there is more to learn right here.

- SEQ_GP_Button then checks to see if a “callback hook” has been installed. I am not 100% clear on the function of the callback hook, but if it is not installed the application will end up branching to either Step edit mode or song mode.
- If the hook has been installed than the application branches to CS_MENU_ExecMenuGPCallbackHook [502]
- Once the application has branched to this point the program flow becomes harder to read, so we are going to start with a higher level analysis of how the menus are mapped and how parameters are accessed, edited and stored.
- Those who have worked with modern GUI programming are probably familiar with the event driven model for creating an interface. At this point it is unclear to what extent the SEQ's menu's can be considered event driven, but it is probably best not to try squeezing them into that mold as there are some important differences.
- The menus are implemented through a table of function pointers that are retrived through an index which is determined by the context of the mode and GP button number. You can see the menu tables in cs_menu_tables.inc

Spend some time reading that file until you get a grasp of it's premise. Menus are created and navigated by indexing through a matrix of function pointers. Take note of some parameters from cs_menu.inc:

- CS_MENU_CURSOR_POS

Let's allow that to sink in for awhile.

- Next we want to know how parameters are modified. From the operation of the SEQ we know that the navigation functions (Right, Left, Datawheel, Select, Exit) are used to select a parameter within a menu and then the Datawheel and Right/Left buttons can increment it's value. Browsing through cs_menu.inc we can find the following three functions which sum up these operations:
  - CS_MENU_Left [313]
  - CS_MENU_Right [320]
  - CS_MENU_Enc [328]
- Each one of these functions places a value in WREG (CS_MENU_Enc already has the encoder direction in WREG) and then calls CS_MENU_AddMenuParameter [573]. From the comments:

```
;; This function adds WREG to CS_MENU_PARAMETER_[LH] and saturates ;; if the max value has been reached ;; IN: add value in WREG ;; OUT: result in CS_MENU_PARAMETER_L ;; branches to menu parameter hook and requests screen update ;;
```