Foenix Rising

A newsletter for developers, users, and enthusiasts of Foenix Retro Systems products

(DRAFT !)

This MIDI focused 9-page issue is sure to please. We examine requisite details and then present 5 pages of code and instruction to help get your legacy F256 platform *off* the desktop and *into* the music studio. Topics include serial communication, a Dream SAM2695 IC, use of a Vishay Optocoupler for MIDI IN, and code to tie it together.

This issue of Foenix Rising can be thought of as the 'missing manual' to uplift your system with new capabilities and purpose. All you need are basic DIY/soldering skills, $25 in parts, and the desire to achieve something you might not have thought possible.

July / August
2024
Issue F16
* **DRAFT** *

General MIDI for the F256K and F256 Jr. - Wavetable synthesis on a budget

The Basics of MIDI communication

The MIDI protocol moves data between a '*sending'* host [computer or controller] and *'listening'* [synth or drum machine] devices at 31,250 bits per second. If you have experience with serial communications, you might think of this as a non-standard or unusual data rate, and it is.

It's hard to pinpoint the history, but certainly, a 1 MHz. clock source divided by 32 is indeed 31.25K. You won't be able to convince your old modem (or most vintage computers) to comply with this rate, but as luck has it, the serial communication block of of your F256 VICKY allows **complete** flexibility of clock values; it's just a matter of working out the math and storing values into a few registers. Otherwise, familiar serial protocol concepts apply such as 'data length' (8 bits),'parity' (none), and 'stop bits' (1).

This month, we have two F256 DIY projects:

#1: General MIDI circuit integration and **#2**: a MIDI IN interface based on an 8-bit ATARI project*; each uses the Feather footprint of your F256. You need not tackle both projects, but it's far more satisfying if you do.

In a future issue, we'll discuss and provide plans for an 'external' (DB9) based solution that adds MIDI OUT and a PIC driven message filter that can be used to sync analog synthesizer gating or clocking for an arpeggiator.

The DREAM SAM2695 IC - it's so 'in' this season

DREAM is a French company (dream.fr) specializing in synthesizer and DSP ICs and their technology has been part of the PC gaming scene for years.

Their SAM2695 product is a QFN48 (48-pin surface mount IC) that you may already be familiar with. It appeared on wavetable daughter boards (as the "S2" in the 2017/2018 time frame) and is still available for sale today. It can also be found in the Lotharek JIL SAMMER for 8-bit ATARI platforms, and numerous DIY solutions.

Early this year, Kevin Williams (TexElec) announced a pair of UART based serial cards for the Commander X16; one with an Espressif ESP32 WiFi adapter and another featuring MIDI ports and a DREAM IC onboard. The WiFi card began shipping in mid-May; release of the MIDI DREAM card is delayed, but expected soon.

Also coming soon to a F256K2 near you …

Foenix fans will be happy to learn that Stefany Allaire will shortly release the Foenix F256K2 system and it has a SAM2695 and optional MIDI DIN jacks onboard. This is in addition to the YAMAHA OPL3, and a pair of SID and PSG instances in FPGA. Pre-order boards for early adopters are expected to be delivered in the September or October time frame with production runs to follow.

Prepare to be impressed - Click here for a 6-minute demo of a SAM2695. The is the JIL SAMMER, captured in direct stereo from an ATARI 800XL; this is the same IC that is the subject of this article. (yes, your F256 will sound THIS good !!)

The remainder of this article focuses on a solution for existing F256 owners, one that you can leverage today. As a DIY solution, this would not be possible without an affordable and easily adaptable product from 'M5stack'; we'll be dissecting their 'SYNTH' product.

M5stack is a Chinese company that manufactures relatively low-cost building block packaged electronics components for educational and prototyping purposes. Several of them have low-speed TTL UART front-ends and are therefore ripe for use with your F256.

The M5stack "SYNTH" product includes most of what we need and can be procured for as little as $12.95 USD plus tax and shipping.

Removal of a 1.5mm hex screw allows splitting of a plastic shell, exposing the tiny DREAM IC and a handful of components along with a tiny IC amp which powers a low-budget one watt, 8-ohm speaker.

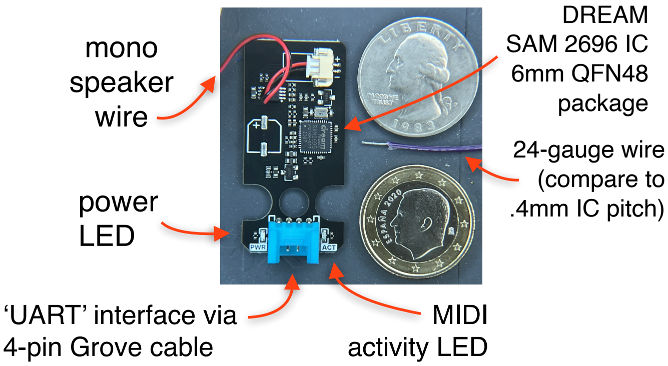
Of note: M5stack was acquired by Espressiv (maker of ESP8266, ESP32) in April. One look and you'll know why.

* Thank you to Michael St. Pierre for sharing his work and fielding questions on his ATARI MIDI interface design

Anatomy of the M5stack SYNTH module

The picture below details internals of the module. On a small board, you'll find a 4-pin connector which brings +5, ground, and MIDI IN to the device (the 4th wire is not used). The unit is complete with a 12 MHz. timing crystal, a 3.3V voltage regulator, and reset circuitry.

The output of a NS4150B amplifier brings left and right DREAM audio to a single mono signal and a short length of 26 gauge wire connects to the SQ-2030 style speaker, attached to the top shell of the molded plastic case with adhesive (easily removed). This is the basis for the first of our two projects.


mono speaker wire
DREAM SAM 2696 IC 6mm QFN48 package
24-gauge wire (compare to .4mm IC pitch)
power LED
PWR
ACT
'UART' interface via 4-pin Grove cable
MIDI activity LED

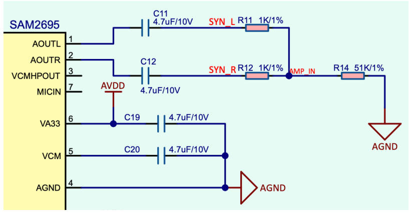
TTL vs. RS-232 voltage signals

Most of the M5stack products are geared for ESP8266/32 or RasPi projects and as a result, are designed for TTL (transistor-to-transistor logic) signals. This is great for an F256 'internal' solution, but we will need to make adjustments to safely use this with DIN based MIDI devices (keyboard controllers, drum machines, or hardware based sequencers and pads).

This can be accomplished with a simple (and single) MOSFET transistor along with a few pull-up resistors, with an old school Ti 232MAX IC and a handful of 1uF capacitors, *or*, a simple optocoupler circuit. All of these solutions are off-the-shelf and inexpensive and can be procured through Amazon or electronic parts houses such as Digikey or Mouser in North America. For the SYNTH build part of the project, we don't need any parts whatsoever; for the MIDI IN interface, we'll use a Vishay 4N28 optocoupler (see BOM on pg. 8).

There is stereo in there… somewhere

The SYNTH product was designed for educational purposes. It provides an interface sufficient for instant-gratification-type projects, but it does not bring the left and right stereo channels to output. I'm not a big fan of their choice of the NS4150B amplifier, but it will work assuming we keep tabs on output level[1].

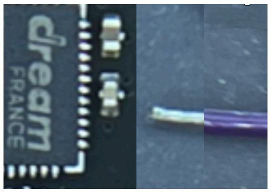


On the schematic, we see that left and right audio channels are combined to mono. This will suffice for amusement or development but eventually, we'll want instruments panned in a stereo field to ideally, use the reverb and echo effects that the SAM2695 has to offer.

How do we go about modding the M5Stack for stereo? The good news is, a schematic is provided. The bad news, the bottom board traces are not exposed and worse, for my eyesight, the scale is impossibly small.

In my video on the subject, I mention that I purchased five of these so I could fry 3 of them in the process of trying to split mono to stereo. I'll share my progress. Survivors will be gifted, so ask me if you are interested.

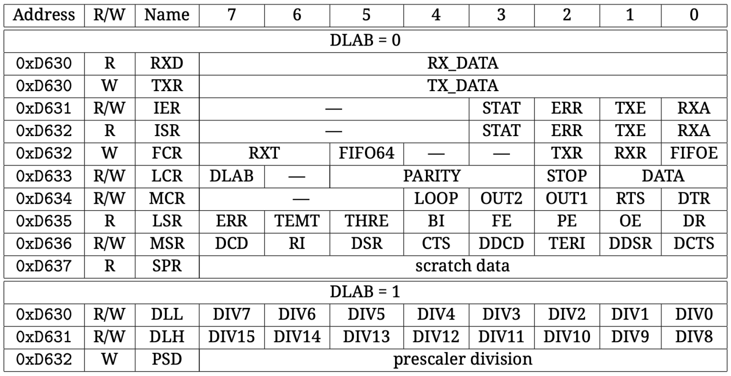
The pic on the left shows the device next to a U.S. quarter and a 1 Euro coin. This picture -> conveys the challenge; it's an extreme closeup of 24 gauge wire versus one side of the SAM2695 IC (12 contact points per side).



Ultimately, we will seek to tap into a resistor or capacitor point. Doing so will require a study of the SAM2695 spec sheet against the M5stack schematic. It will also destroy any hope of the warranty (surprise!).

Software details - COMM setup and our first tone

The F256 includes a 16750 UART core in TINY VICKY which is configured via 8 registers. Because they are multiplexed, we need to manipulate something called the DLAB bit for some features. But once the clock divisor is set, we will not need to touch DLAB again. For ref, here is the register map from the F256 manual:



Address	R/W	Name	7	6	5	4	3	2	1	0
DLAB = 0										
0xD630	R	RXD	RX_DATA							
0xD630	W	TXR	TX_DATA							
0xD631	R/W	IER	—				STAT	ERR	TXE	RXA
0xD632	R	ISR	—				STAT	ERR	TXE	RXA
0xD632	W	FCR	RXT		FIFO64	—	—	TXR	RXR	FIFOE
0xD633	R/W	LCR	DLAB	—	PARITY			STOP	DATA	
0xD634	R/W	MCR	—			LOOP	OUT2	OUT1	RTS	DTR
0xD635	R	LSR	ERR	TEMT	THRE	BI	FE	PE	OE	DR
0xD636	R/W	MSR	DCD	RI	DSR	CTS	DDCD	TERI	DDSR	DCTS
0xD637	R	SPR	scratch data							
DLAB = 1										
0xD630	R/W	DLL	DIV7	DIV6	DIV5	DIV4	DIV3	DIV2	DIV1	DIV0
0xD631	R/W	DLH	DIV15	DIV14	DIV13	DIV12	DIV11	DIV10	DIV9	DIV8
0xD632	W	PSD	prescaler division							

Code to init the serial interface

The following procedure is necessary regardless of Foenix model (F256K or F256 Jr.) and is identical for internal (ESP 8266 socket) or external (DB9 serial) use. As SuperBASIC is without **OPEN** and **CLOSE** device management verbs, we will need to use **POKE**.

This following SuperBASIC code begins by setting the MMU_IO_CTRL register to 0 in order to expose the I/O pages to the $C000-$DFFF address range.

With this accomplished, we store values into registers and in just 5 lines[2], the UART is configured and we are ready to send MIDI messages.

See (1), (2): *liner notes* on pg. 9 for more on these topics

```
poke 1,0              set mmu to i/o page
poke $D633,128        set DLAB to 1 to expose
                      bps rate in $d630-$d632
poke $D630,50         bps clock divisor (low)
poke $D631,0          bps clock divisor (high)
poke $D632,0          bps prescaler divisor
poke $D633,3          no parity, 1 stop bit, 8 bit word
                      clear DLAB
```

That's all there is to it. The following 65C02 assembly language code accomplishes the same task:

```
STZ   $01
LDA   #$80
STA   $D633
LDA   #$32
STA   $D630
STZ   $D631
STZ   $D632
LDA   #$03
STA   $D633
```

Two subtle nuances pertaining to working with the 16550/16750 that may not appear obvious at first glance:

a. A handful of its register are dual purpose, depending on read (R) or write (W) and some have a third purpose when DLAB comes into the picture. For example, $D630 is used to SEND a byte, RECV a byte, and [in DLAB mode] prescribes the lower 8 bits of the clock divisor. Keep this in mind.

b. Outside of bps timing, address $D633 is bound to be the most important register and an area that will keep you guessing when things are going wrong. Here is an explanation of how we get to a value of 3.

   This value prescribes an 8 bit word with 1 stop bit and no parity. And since we leave DLAB with a 0 bit value, we escape the clock divisor setting mode and ready the 16750 for writes (and potentially reads) from $D630. Absolutely everything else we will ever need to do to communicate with MIDI devices is accomplished by poking bytes into $D630.

   Here are excerpts from the F256 manual, laid out bitwise (horizontally) rather than in consecutive tables. The yellow highlights identify values used. (bit 6 is not implemented)

7	6
DLAB	–

LCR5	LCR4	LCR3	Parity
—	—	0	NONE
0	0	1	ODD
0	1	1	EVEN
1	0	1	MARK
1	1	1	SPACE

LCR2	Stop Bits
0	1
1	1.5 or 2

LCR1	LCR0	Length
0	0	5
0	1	6
1	0	7
1	1	8

Of course, the 16750 has additional registers that we do not need. They implicate buffer and FIFO behavior and status/error conditions. And some pertain to signals which are not implemented in TINY VICKY and thus, are not available to F256 platforms. If you've worked with serial communications prior, some of these will sound familiar: Data Terminal Ready (DTR), Clear to Send (CTS), Request to Send, Carrier Detect (DCD).

The closing point is, depending on your application, you may need to alter FIFO settings, but for simple MIDI testing, the setup above will suffice. Once interfaced, amazing sound is just a few lines of code away.

Hidden talents and some simple code to play a note

Before we command the SYNTH to play its first tone, there is something else to talk about. We've discussed that the DREAM IC is inexpensive and since it's GM compliant, it's powerful. You are probably also aware that it more than doubles the voice requirements of GM (boasting up to 64 voices of polyphony). And of course, it also has a wonderfully derived traditional, orchestral, synthesized, and percussive instruments.

But wait, there's more… Did you know that the DREAM chip includes a full featured effects engine (enabled with MIDI NRPN commands) to affect stereo panning, frequency filtering, chorus and reverb, just to name a handful.

Leveraging this capability will reduce polyphony to 38, but has use cases that are beyond the obvious. Imagine heavily processed haunting background music or growling wind effects growing louder and stronger as you descend into the lower levels of Micah Bly's *Lair of the Lich King*. The addition of sonic cues will add new dimensions to game play and developers will be relieved from the struggles of coding for SID ICs or being stuck with plain vanilla PSG square waves.

The DREAM IC also has a MIC input line which can be processed through Echo, EQ, and spacial effects. Unfortunately, the tiny SYNTH module that we are experimenting with does *not* implement pin 7. But with this said, there is still plenty to experiment with. (read more on these features on pg. 15 of this spec sheet)

Back to our regularly scheduled programming…

The following is little less interesting, but it's a start. It's a 'hello world' MIDI example. We follow this up with something more interesting: a demo of all 128 patches followed by a run of the GM percussion instruments from channel 10, program 10. (programs are named MIDINOTES.BAS and MIDIPERC.BAS and can be downloaded here)

This buildup leads us to an a-ha moment; Unlike traditional audio ICs which require knowledge of synthesis plus programatic control of envelopes and LFOs to be convincing, wavetable based GM takes care of the difficult part for us. Pianos, trumpets, synths, and percussion sound as they should. They are based on sampled waveform partials, have natural sounding loop points, instrument appropriate envelopes, vibrato where it should be, and often, the ambiance of the original instruments. You'd suffer an inguinal hernia injury trying to accomplish a fraction of this on a SID IC.

I'll be contributing a MIDI monitor with a hex string 'send' utility (for NRPN or traditional messages), but my every-burning hope is for others to get involved. A head start will be directly applicable to the F256K2.

How to: play a *single note* - 4th octave C on channel 1 | How to: send a program change (select an instrument)

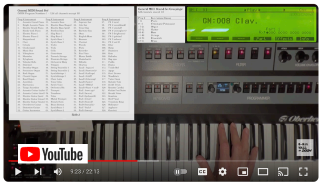
Following initialization of the serial interface (above), try the following two SuperBASIC examples:

```
poke $D630,144              note on message
poke $D630,60               note number
poke $D630,127              note velocity 127=max

for x=1 to 1000: next       note length delay

poke $D630,128              note off command
poke $D630,60               note number
poke $D630,127              release velocity
```

```
poke $D630,192              program change message
poke $D630,{program #}      instrument # 0..127
```



… now re-execute the note code and hear the difference !!

For a primer on GM programs (patches), by group/family, click here

A note about instrument selection: without a program change, this example will play using program #1 (MIDI 0), which is a “Grand Piano”. It is important to understand that some General MIDI instruments (the Piano is one) have a ‘natural’ decay/release where a note-on command executed without an accompanying note-off, will eventually fade to zero volume. In the above example, we pause for a 1000 count in a for/next loop. Feel free to alter this value and assess the difference in sonic dynamics (the same applies to note velocity which may change timbre in addition to max. volume).

Instruments such as “Church Organ” on the other hand, sustain indefinitely and upon note-off, release near immediately. Characteristics of some instruments (tuned by DREAM engineers), require longer interval between note-on and note-off.

Our first example code, MIDINOTES.BAS, runs the gamut through all 128 patches, playing notes C-D-E in sequence with brief pauses between notes and program changes. Of course, patches with longer attack and decay cycles may not sound quite right. These programs will benefit from a longer note length, however, this demo program was constructed simply. Feel free to experiment, altering timing to discover more about the way specific instruments are voiced.

Using a MIDI keyboard attached to your F256 with example #4 below is the best way to experiment. You’ll need the 2nd project which adds MIDI IN (and mentioned above, a simple ‘monitor’ program will be published shortly).

Getting serious - programs* to try; ripe for modification

Example #1: MIDINOTES.BAS (plays 3 notes on each of 128 instruments, pausing between; on/off; velocity = 127)

```
10     rem "vars; pgm=program (patch #); nlen=note_length; plen=pause_length"
20     pgm=0:nlen=500:plen=1000

30     rem "-------------- lines 40 through 90 setup bps"
40     poke 1,0:rem "      set io_ctrl to i/o page"
50     poke $D633,128:rem "set dlab to 1 to expose bps rate in $d630-$d632"
60     poke $D630,50:rem " bps clock divisor (low byte)"
70     poke $D631,0:rem "  bps clock divisor (high byte)"
80     poke $D632,0:rem "  prescale divisor (for bps)"
90     poke $D633,3:rem "  no parity/1 stop bit/8 bit word and clear dlab”

100    rem "-------------- lines 200 through 255 play notes"
200    poke $D630,144:poke $D630,60:poke $D630,127:rem "note c4 (on)"
205    for x=1 to nlen:next
210    poke $D630,128:poke $D630,60:poke $D630,127
215    for x=1 to plen:next
220    poke $D630,144:poke $D630,62:poke $D630,127:rem "note d4 (on)"
225    for x=1 to nlen:next
230    poke $D630,128:poke $D630,62:poke $D630,127
235    for x=1 to plen:next
240    poke $D630,144:poke $D630,64:poke $D630,127:rem "note e4 (on)"
245    for x=1 to nlen:next
250    poke $D630,128:poke $D630,64:poke $D630,127
255    for x=1 to plen:next

260    rem "-------------- the following calls pgm inc and loops unless 128"
265    gosub 300:if pgm=128 then goto 400
270    goto 200
275    rem "------------------------------------"
300    poke $D630,192:pgm=pgm+1:poke $D630,pgm
305    return
400    end
```

[*] Prefer to skip the typing? Download the examples from the Foenix Content Marketplace at http://apps.emwhite.org/foenixmarketplace. And for a quick byte-by-byte primer on the MIDI and General MIDI command structure, check out my 8-Bit Wall of Doom YouTube series

Example #2: MIDIPERC.BAS **(3 hits per percussive instrument #35-82 at velocity levels 40, 80, and 127; channel 10)**

```
10     rem "vars; note=note aka instr; nlen=note length; plen=pause length"
20     note=35:nlen=500:plen=1000

30     rem "-------------- lines 40 through 90 setup bps"
40     poke 1,0:rem "       set io_ctrl to i/o page"
50     poke $D633,128:rem "set dlab to 1 to expose bps rate in $d630-$d632"
60     poke $D630,50:rem " bps clock divisor (low byte)"
70     poke $D631,0:rem "  bps clock divisor (high byte)"
80     poke $D632,0:rem "  prescale divisor (for bps)"
90     poke $D633,3:rem "  no parity/1 stop bit/8 bit word and clear dlab”

100    rem "-------------- lines 200 through 255 play notes"
110    poke $D630,192+9:poke $D630,9   : rem "set instrument to patch 10"
200    poke $D630,144+9:poke $D630,note: poke $D630,40
205    for x=1 to nlen:next
210    poke $D630,128+9:poke $D630,note: poke $D630,127
215    for x=1 to plen:next
220    poke $D630,144+9:poke $D630,note: poke $D630,80
225    for x=1 to nlen:next
230    poke $D630,128+9:poke $D630,note: poke $D630,127
235    for x=1 to plen:next
240    poke $D630,144+9:poke $D630,note: poke $D630,127
245    for x=1 to nlen:next
250    poke $D630,128+9:poke $D630,note: poke $D630,127
255    for x=1 to plen:next

260    rem "-------------- the following calls pgm inc and loops unless 82"
265    gosub 300:if note=82 then goto 400
270    goto 200

275    rem "--------------------------------------"
300    note=note+1
305    return
400    end
```

*NOTE: add lines 40-90 to example #3 below

Will any of this work on the F256K2? Of course!

Simply omit lines 30-90. The only req'd change is the MIDI out register address (serial TX_DATA on the F256 or $D630)

Example #3*: **Real-time message filtering & channel 1 sequencer output mapped to GM percussion channel 10**

In this program, embedded assembly language is leveraged to accomplish something that requires more horsepower.

The endless loop below performs rudimentary filtering. It reads from MIDI IN, discards real-time messages, transposes NOTE ON messages to channel 10 (percussion), and sends the resulting byte stream to our SYNTH device. This was written to integrate with a Roland TR-08/TR-09 drum machine, but will work with others. Check out this video.

Program change to 10 (percussion)

Enables 64-byte FIFO and ‘polling’ mode (not interrupt driven)
Bit 0 of $D635 = 1 when a byte is pending (see 170-190 below)

```
100 poke $D632,231: poke $631,0
110 poke $D630,192: poke $D630,9
120 ml_routines(): call loop
130  proc ml_routines()
140   mlroutines=$7C00
150   for c=0 to 1
160     assemble mlroutines,c

170     .loop   lda $D635
180             and #$01
190             beq loop
200             lda $D630
210             inc $01
220             inc $01
230             inc $C000
240             sta $C000,x
250             stz $01
260                 cmp #$FE
270                 beq loop
280                 cmp #$F8
290                 beq loop
300                 cmp #$90
310                 beq noteon
320                 bra sendbyte

330 .noteon     lda #$99
340 .sendbyte   sta $D630
350             inx
360             bra loop
370    next
380 endproc
```

polling test (lines 170-190)

These 5 lines switch the I/O bank to TEXT then back to registers (lines 210-250)

write byte to screen for visual indication of message processing (line 240)

discard MIDI clock sense (line 260)

discard MIDI clock sync (line 280)

If note on msg. (channel 1) (line 300)

remap to channel 10 (percussion) (line 330)

Example #4: **MIDI message loopback for keyboard playing (taking example #3 two steps further)**

This complete program lets you 'play' your Foenix as if it was a MIDI synthesizer module. Most of the time, the code simply reads a byte (from MIDI IN) and sends it to the SYNTH device as is. But it does have three features:

a) In normal mode, input from MIDI IN (received on the 'TX' pin of the Feather 8229 footprint) is echoed to the 'RX' pin of the Feather footprint (which is expected to be wired to SYNTH). Messages are displayed as above but NOTE ON, NOTE OFF, PROGRAM CHANGE, and REAL-TIME message bytes are rendered in color. All other messages are displayed in GREY text; this includes CHANNEL PRESSURE, PITCH BEND, MOD WHEEL, etc.

b) By default, there is no channel remapping (all incoming data is sent directly to the SYNTH). If incoming data is polluted with real-time messages, these bytes will be displayed in RED text.

c) Three 'gestures' support mode toggle and program change (for channel 1 only), however, you may use your MIDI controller to configure multiple of the 16 channels for different programs/patches to play different voices concurrently. Here is an overview of how gestures are interpreted:

- Playing an octave 4 [C, C#, D, D#, E] 'chord' (awkward for a reason) will toggle between the normal pass-thru mode and percussion mode (channel 10/program 10). Listen for a sequence of notes, played upon mode switch.
- While in normal mode, the same key sequence in octave 5 increments the program number by 1; and in octave 6, moves to patch 1 of the next group. There is no audible indication until you play a key. Wrap around applies to both of these functions. If your keyboard only supports one or two octaves, you will need to use the octave control built into your keyboard in order to align the key/MIDI notes as required. The 'online' version of this code will shortly be updated to display the GM program/patch name per channel and the operating mode.

The remaining 3.5 pgs of this article are in draft and will be posted in the coming week