emfbadge16

Claude Heiland-Allen

2017

Contents

1 bytebeat/0001-enable-pyb.DAC-digital-to-analog-convertor-api.patch

```
From 8bed55122b601e530c8e4e9d12467323d0b5a6fa Mon Sep 17 00:00:00 2001
From: Claude Heiland-Allen <claude@mathr.co.uk>
Date: Thu, 2 Mar 2017 12:55:20 +0000
Subject: [PATCH 1/2] enable pyb.DAC digital to analog convertor api

---
 stmhal/boards/STM32L475_EMFBADGE/mpconfigboard.h | 1 +
 1 file changed, 1 insertion(+)

diff --git a/stmhal/boards/STM32L475_EMFBADGE/mpconfigboard.h b/stmhal/boards/↙
    ↪ STM32L475_EMFBADGE/mpconfigboard.h
index be029221..6fa81ed1 100644
--- a/stmhal/boards/STM32L475_EMFBADGE/mpconfigboard.h
+++ b/stmhal/boards/STM32L475_EMFBADGE/mpconfigboard.h
@@ -12,6 +12,7 @@
 #define MICROPY_HW_ENABLE_RTC       (1)
 #define MICROPY_HW_ENABLE_TIMER     (1)
 #define MICROPY_HW_ENABLE_SERVO     (1)
+#define MICROPY_HW_ENABLE_DAC       (1)
 #define MICROPY_HW_HAS_UGFX         (1)
 #define MICROPY_HW_HAS_CC3100       (1)
 #define MICROPY_HW_L4_512_FS        (1)
--
2.11.0
```

2 bytebeat/0002-Revert-stmhal-dac-DAC-deinit-method-added.patch

```
From 04bbba79308f93c3db87de23e3646b1da0496041 Mon Sep 17 00:00:00 2001
From: Claude Heiland-Allen <claude@mathr.co.uk>
Date: Thu, 2 Mar 2017 13:28:24 +0000
Subject: [PATCH 2/2] Revert "stmhal/dac: DAC deinit() method added."

This reverts commit 641300dccb0a68e4ede07323aabeed564a892a8f.
---
 docs/library/pyb.DAC.rst |  4 ----
```

```
 stmhal/dac.c              | 16 ----------------
 2 files changed, 20 deletions(-)

diff --git a/docs/library/pyb.DAC.rst b/docs/library/pyb.DAC.rst
index 4b9651e2..f0d4de4c 100644
--- a/docs/library/pyb.DAC.rst
+++ b/docs/library/pyb.DAC.rst
@@ -67,10 +67,6 @@ Methods

     Reinitialise the DAC.  ``bits`` can be 8 or 12.

-.. method:: DAC.deinit()
-
-   De-initialise the DAC making its pin available for other uses.
-
 .. method:: DAC.noise(freq)

     Generate a pseudo-random noise signal.  A new random sample is written
diff --git a/stmhal/dac.c b/stmhal/dac.c
index 7493bb59..bb700883 100644
--- a/stmhal/dac.c
+++ b/stmhal/dac.c
@@ -246,21 +246,6 @@ STATIC mp_obj_t pyb_dac_init(mp_uint_t n_args, const ↙
    ↳ mp_obj_t *args, mp_map_t *k
 }
 STATIC MP_DEFINE_CONST_FUN_OBJ_KW(pyb_dac_init_obj, 1, pyb_dac_init);

-/// \method deinit()
-/// Turn off the DAC, enable other use of pin.
-STATIC mp_obj_t pyb_dac_deinit(mp_obj_t self_in) {
-    pyb_dac_obj_t *self = self_in;
-    if (self->dac_channel == DAC_CHANNEL_1) {
-        DAC_Handle.Instance->CR &= ~DAC_CR_EN1;
-        DAC_Handle.Instance->CR |= DAC_CR_BOFF1;
-    } else {
-        DAC_Handle.Instance->CR &= ~DAC_CR_EN2;
-        DAC_Handle.Instance->CR |= DAC_CR_BOFF2;
-    }
-    return mp_const_none;
-}
-STATIC MP_DEFINE_CONST_FUN_OBJ_1(pyb_dac_deinit_obj, pyb_dac_deinit);
-
 #if defined(TIM6)
 /// \method noise(freq)
 /// Generate a pseudo-random noise signal.  A new random sample is written
@@ -476,7 +461,6 @@ STATIC MP_DEFINE_CONST_FUN_OBJ_KW(pyb_dac_write_timed_obj, ↙
    ↳ 1, pyb_dac_write_time
 STATIC const mp_map_elem_t pyb_dac_locals_dict_table[] = {
     // instance methods
     { MP_OBJ_NEW_QSTR(MP_QSTR_init), (mp_obj_t)&pyb_dac_init_obj },
-    { MP_OBJ_NEW_QSTR(MP_QSTR_deinit), (mp_obj_t)&pyb_dac_deinit_obj },
     { MP_OBJ_NEW_QSTR(MP_QSTR_write), (mp_obj_t)&pyb_dac_write_obj },
     #if defined(TIM6)
     { MP_OBJ_NEW_QSTR(MP_QSTR_noise), (mp_obj_t)&pyb_dac_noise_obj },
--
2.11.0
```

3 bytebeat/main.py

```
### Author: Claude Heiland-Allen
### Description: Live coding byte beat synth
### Category: Music
### License: MIT
### Appname : ByteBeat

import buttons
import imu
import ugfx
import pyb
import array
import math

code = ([">>", "&", "f", "t", "t"], [])

buttons.init()
accel = imu.IMU()
ugfx.init()

ugfx.clear(ugfx.BLACK)
ugfx.set_default_font(ugfx.FONT_TITLE)
s = ugfx.Style()
s.set_enabled([ugfx.WHITE, ugfx.WHITE, ugfx.WHITE, ugfx.WHITE])
s.set_pressed([ugfx.WHITE, ugfx.WHITE, ugfx.WHITE, ugfx.WHITE])
s.set_disabled([ugfx.WHITE, ugfx.WHITE, ugfx.WHITE, ugfx.WHITE])
s.set_focus(ugfx.BLACK)
s.set_background(ugfx.BLACK)
ugfx.set_default_style(s)

bytecode = array.array('i', [])

opcodes = { "~": 0x100, "<": 0x200, "=": 0x300, ">": 0x400,  "%": 0x500, "/": 0x600, "*": 0x700, "t": 0x800,  "+": 0xa00, "-": 0xb00, "<<": 0xc00,  ">>": 0xd00, "^": 0xe00, "|": 0xf00, "&": 0x1000 }

@micropython.asm_thumb
def interpret_asm(r0, r1, r2): # t, bytecode array, bytecode array length
        # save stack
        push({r10})
        mov(r10, r13)
        # clear stack underflow safety buffer
        mov(r3, 0)
        push({r3})
        push({r3})
        push({r3})
        push({r3})
        push({r3})
        push({r3})
        push({r3})
        push({r3})
        push({r3})
        push({r3})
        push({r3})
        push({r3})
        push({r3})
        # for r3 in [0..length)
```

```
        cmp(r3, r2)
        blt(LOOP)
        b(DONE)
        label(LOOP)

        ldr(r4, [r1, 0])
        # r4 contains bytecode (2 bytes)
        add(r1, r1, 4)
        mov(r5, r4)
        mov(r6, 8)
        lsr(r5, r6)
        # r5 contains opcode flag

        # ~
        cmp(r5, 0x1)
        itttt(eq)
        pop({r6})
        mvn(r6, r6)
        push({r6})
        b(NEXT)
        # <
        cmp(r5, 0x2)
        bne(SKIP2)
        pop({r6, r7})
        cmp(r7, r6)
        ite(lt)
        mov(r6, 1)
        mov(r6, 0)
        push({r6})
        b(NEXT)
        label(SKIP2)
        # ==
        cmp(r5, 0x3)
        bne(SKIP3)
        pop({r6, r7})
        cmp(r6, r7)
        ite(eq)
        mov(r6, 1)
        mov(r6, 0)
        push({r6})
        b(NEXT)
        label(SKIP3)
        # >
        cmp(r5, 0x4)
        bne(SKIP4)
        pop({r6, r7})
        cmp(r7, r6)
        ite(gt)
        mov(r6, 1)
        mov(r6, 0)
        push({r6})
        b(NEXT)
        label(SKIP4)
        # %
        # no modulo instruction, so do a % b = a - (a / b) * b
        cmp(r5, 0x5)
        bne(SKIP5)
```

```
        pop({r6, r7})
        cmp(r6, 0)
        it(eq)
        mov(r6, 1)
        sdiv(r5, r7, r6) # safe to use r5 as we'll branch to NEXT
        mul(r5, r6) # look into mls() instruction?
        sub(r7, r7, r5)
        push({r7})
        b(NEXT)
        label(SKIP5)
        # /
        cmp(r5, 0x6)
        bne(SKIP6)
        pop({r6, r7})
        cmp(r6, 0)
        it(eq)
        mov(r6, 1)
        sdiv(r7, r7, r6)
        push({r7})
        b(NEXT)
        label(SKIP6)
        # *
        cmp(r5, 0x7)
        itttt(eq)
        pop({r6, r7})
        mul(r6, r7)
        push({r6})
        b(NEXT)
        # t
        cmp(r5, 0x8)
        itt(eq)
        push({r0})
        b(NEXT)
        # 0
        cmp(r5, 0x9)
        itttt(eq)
        mov(r6, 0xFF)
        and_(r4, r6)
        push({r4})
        b(NEXT)
        # +
        cmp(r5, 0xA)
        itttt(eq)
        pop({r6, r7})
        add(r6, r6, r7)
        push({r6})
        b(NEXT)
        # -
        cmp(r5, 0xB)
        itttt(eq)
        pop({r6, r7})
        sub(r7, r7, r6)
        push({r7})
        b(NEXT)
        # <<
        cmp(r5, 0xC)
        itttt(eq)
```

```
            pop({r6, r7})
            lsl(r7, r6)
            push({r7})
            b(NEXT)
            # >>
            cmp(r5, 0xD)
            itttt(eq)
            pop({r6, r7})
            asr(r7, r6)
            push({r7})
            b(NEXT)
            # ^
            cmp(r5, 0xE)
            itttt(eq)
            pop({r6, r7})
            eor(r6, r7)
            push({r6})
            b(NEXT)
            # |
            cmp(r5, 0xF)
            itttt(eq)
            pop({r6, r7})
            orr(r6, r7)
            push({r6})
            b(NEXT)
            # &
            cmp(r5, 0x10)
            itttt(eq)
            pop({r6, r7})
            and_(r6, r7)
            push({r6})

            # done
            label(NEXT)
            add(r3, r3, 1)
            cmp(r3, r2)
            blt(LOOP)
            label(DONE)
            pop({r0})
            mov(r3, 0xFF)
            and_(r0, r3)
            # restore stack
            mov(r13, r10)
            pop({r10})

    def interpret(bytecode, t):
            # pad with 16x 0 in case of underflow...
            stack = [0,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0]
            n = len(bytecode)
            i = 0
            while i < n:
                    b = bytecode[i]
                    o = b >> 8
                    if   o == 1:
                            stack = [~stack[0]] + stack[1:]
                    elif o == 2:
                            stack = [int(stack[1] < stack[0])] + stack[2:]
```

```
                elif o == 3:
                        stack = [int(stack[1] == stack[0])] + stack[2:]
                elif o == 4:
                        stack = [int(stack[1] > stack[0])] + stack[2:]
                elif o == 5:
                        if stack[0] == 0:
                                stack = [stack[1] % 1] + stack[2:]
                        else:
                                stack = [stack[1] % stack[0]] + stack[2:]
                elif o == 6:
                        if stack[0] == 0:
                                stack = [stack[1] / 1] + stack[2:]
                        else:
                                stack = [stack[1] / stack[0]] + stack[2:]
                elif o == 7:
                        stack = [stack[1] * stack[0]] + stack[2:]
                elif o == 8:
                        stack = [t] + stack
                elif o == 9:
                        stack = [b & 0xFF] + stack
                elif o == 0xa:
                        stack = [stack[1] + stack[0]] + stack[2:]
                elif o == 0xb:
                        stack = [stack[1] - stack[0]] + stack[2:]
                elif o == 0xc:
                        stack = [stack[1] << stack[0]] + stack[2:]
                elif o == 0xd:
                        stack = [stack[1] >> stack[0]] + stack[2:]
                elif o == 0xe:
                        stack = [stack[1] ^ stack[0]] + stack[2:]
                elif o == 0xf:
                        stack = [stack[1] | stack[0]] + stack[2:]
                elif o == 0x10:
                        stack = [stack[1] & stack[0]] + stack[2:]
                i = i + 1
        return stack[0] & 0xff

# double buffering
t = 0
which = 0
block = (bytearray(256), bytearray(256))
def interprets():
        global bytecode
        global t
        global block
        global which
        b = block[which]
        n = len(bytecode)
        i = 0
        while i < 256:
                # b[i] = interpret(bytecode, t + i)
                b[i] = interpret_asm(t + i, bytecode, n)
                i = i + 1
        t = (t + 256) & 0x3fffFFFF

dac = pyb.DAC(2)
```

```
def callback(timer):
        global block
        global which
        global dac
        dac.write_timed(block[which], 8192, mode=pyb.DAC.NORMAL)
        which = 1 - which
        interprets()

timer = pyb.Timer(8)
timer.init(freq=32)

def opcode(s):
        global opcodes
        try:
                o = opcodes[s]
        except:
                o = 0x900 | (int("0x" + s) & 0xFF)
        return o


def preprocess(code):
        c = list(reversed(code[0])) + code[1]
        n = len(c)
        i = 0
        while i < n:
                c[i] = opcode(c[i])
                i = i + 1
        return array.array('i', c)

cursor = "@"

def text(code):
        global cursor
        return "\n".join([" ".join(reversed(code[0])), cursor, " ".join(code[1])↙
            ↳ ])

source = ugfx.Label(0, 32, 320, 240 - 64, text(code), justification=ugfx.Label.↙
    ↳ CENTER)
timeout = 0

def update(code):
        global source
        source.text(text(code))

def left():
        global code
        if len(code[0]) > 0:
                code = (code[0][1:], [code[0][0]] + code[1])
                update(code)

def right():
        global code
        if len(code[1]) > 0:
                code = ([code[1][0]] + code[0], code[1][1:])
                update(code)

def delete():
```

```
        global code
        global bytecode
        if len(code[0]) > 0:
                code = (code[0][1:], code[1])
                update(code)
                bytecode = preprocess(code)

choosing = False
inserting = False

cursors = ["~", "<", "=", ">",  "%", "/", "*", "t",  "t", "+", "-", "<<",  ">>",↙
    ↪  "^", "|", "&" ]

def choose():
        global code
        global cursors
        global cursor
        global accel
        a = accel.get_acceleration()
        x = a['x']
        y = a['y']
        r = x * x + y * y
        p = 0.5 + 0.5 * math.atan2(-y, x) / math.pi
        if r > 0.2:
                cursor = hex(int(0xFF * p + 0.5) & 0xFF)[2:]
        else:
                cursor = cursors[int(0x10 * p + 0.5) & 0xF]
        update(code)

def commit(inserting):
        global code
        global cursor
        global bytecode
        if inserting:
                code = ([cursor] + code[0], code[1])
        elif len(code[0]) > 1:
                code = ([cursor] + code[0][1:], code[1])
        cursor = "@"
        update(code)
        bytecode = preprocess(code)

update(code)
bytecode = preprocess(code)
timer.callback(callback)

while not buttons.is_pressed("BTN_MENU"):
        if choosing and not buttons.is_pressed("BTN_A") and not buttons.↙
            ↪ is_pressed("BTN_B"):
                choosing = False
                commit(inserting)
                timeout = 2
        if buttons.is_pressed("BTN_A") and not buttons.is_pressed("BTN_B"):
                choosing = True
                inserting = True
                choose()
        if not buttons.is_pressed("BTN_A") and buttons.is_pressed("BTN_B"):
                choosing = True
```

```
                inserting = False
                choose()
        if timeout == 0:
                if buttons.is_pressed("JOY_LEFT"):
                        left()
                        timeout = 2
                if buttons.is_pressed("JOY_RIGHT"):
                        right()
                        timeout = 2
                if buttons.is_pressed("JOY_CENTER"):
                        delete()
                        timeout = 2
        else:
                timeout = timeout - 1
        t2 = t
        x = (t2 & 0xFF) + 0x20
        y = (t2 >> 8) & 0x1F
        b = block[which]
        ugfx.stream_start(x, y, 256, 1)
        i = 0
        while i < 256:
                o = b[i]
                c = o | (o << 8)
                ugfx.stream_color(c)
                i = i + 1
        ugfx.stream_stop()
        pyb.wfi()
```

4 bytebeat/README.md

```
bytebeat
========

algorithmic symphonies from reverse polish notation

targetting emfcamp 2016 badge tilda mk3


install
-------

needs modified firmware (compiled with dac support), these instructions
modified from those at: https://badge.emfcamp.org/wiki/TiLDA_MK3/build

set up variable to bytebeat sources (directory containing this readme):

    export bytebeat=/path/to/bytebeat/

debian packages for the cross-compiler exist, no need for ppa:

    sudo apt-get install gcc-arm-none-eabi

getting the source:

    git clone --recursive https://github.com/emfcamp/micropython.git
    cd micropython
    git checkout tilda-master
```

```
apply dac-enabling patches:

    git checkout -b enable-dac
    git am $bytebeat/000*.patch

build:

    make -C stmhal BOARD=STM32L475_EMFBADGE

flash to tilde in dfu mode (press joystick center with rear reset button):

    make -C stmhal BOARD=STM32L475_EMFBADGE deploy    # untested by me...

you can also reflash (this method tested by me) with the update.py from
https://badge.emfcamp.org/wiki/TiLDA_MK3/Firmware_Update
but you need to patch it so it doesn't automatically download fresh firmware
from the internet and delete it afterwards:

    cd stmhal/build-STM32L475_EMFBADGE
    wget https://update.badge.emfcamp.org/update.py
    patch -p1 < $bytebeat/update-py*.patch
    python update.py

you need to let the badge connect to wifi and download the base system now.

finally, mount the badge and copy the program:

    export badge=/mnt/badge
    sudo mkdir -p $badge
    sudo mount /dev/sdwhatever $badge
    sudo mkdir -p $badge/apps/claude~bytebeat/
    sudo cp $bytebeat/main.py $badge/apps/claude~bytebeat/main.py

alternatively, you can run with pyboard.py to send over usb one-shot (useful
for development), see:
https://badge.emfcamp.org/wiki/TiLDA_MK3/Get_Started#5._Run_a_whole_file_of_code

controls
--------

joystick left/right to navigate (@ cursor shows current selection for A/B press)

joystick center press to delete previous item

hold button A with accelerometer to insert new items on release

hold button B with accelerometer to modify previous item on release

small tilts choose operators (including t)

large tilts choose values (8bit hex)

menu button to quit
```

```
electronics
-----------

audio out on pin X6 PB2 (8bit, 8kHz, 0V-3.3V), connect to minijack ring
(with 100uF AC coupling capacitor?)

connect X 0V pin to minijack sleeve (with resistor?)

problems with mains hum due to touching pcb ground, consider making a case
```

5 bytebeat/update-py-no-download-and-delete-firmware.patch

```
--- old/update.py       2017-03-02 12:40:47.607835323 +0000
+++ new/update.py       2017-03-05 00:51:30.632207263 +0000
@@ -22,7 +22,6 @@
 import usb.core
 import usb.util
 import zlib
-import urllib
 import os

 # VID/PID
@@ -522,9 +521,6 @@
         init()
         print("We found your badge. So far, so good...")

-        print("Please wait while we download the latest version of the badge software...")
-        urllib.URLopener().retrieve("https://update.badge.emfcamp.org/firmware.dfu", "firmware.dfu")
-        print("Done")
         print("Please wait, we're now running the update")

         elements = read_dfu_file("firmware.dfu")
@@ -543,8 +539,6 @@
         print("To put your badge into DFU mode you need to press the joystick in the middle while pressing the reset button at the back.")
         print()
         print("Error: %s" %(e))
-    finally:
-        if "firmware.dfu" in os.listdir("."): os.remove("firmware.dfu")

 if __name__ == '__main__':
     main()
```