Highlighted item: can only test on rev1

Control

Interfaces by Protocol

I2C

POT_SCL: I2C1_SCL
POT_SDA: I2C1_SDA

(other protocols similar:)
pot_write(): add control data to kernel buffer and enable TxE interrupt
pot_isr(): move data from kernel buffer to i2c data register, clear interrupt
Use circular buffer (producer-consumer), and warn & block when buffer is full.

SPI

ADC_SCK: SPI1_SCK
ADC_MISO: SPI1_MISO
ADC_MOSI: SPI1_MOSI

CLK

MCLK: MCO1
Reference: ref menual 6.2.10 Clock-out capability (p. 158)

I2S

BT_RFS: left/right clock, I2S2_WS
BT_SCLK: bit clock, I2S2_CK
BT_DR: data, I2S2_SD
Reference:
BM83 design guide 3.4 Audio Input to BM83 Source
Ref manual 28.4 I2S functional description

left_data, right_data, left_data, right_data, ...

UART

BT_RXD: USART2_TX
BT_TXD: USART2_RX

Reference:
BM83 host MCU development guide 5.1 Connections Between BM83 and PIC 32 MCU

FSMC (LCD Parallel)

LCD_DB<7:0>: FSMC_D0-7
LCD_CS1B: FSMC_NE1, active low chip select
LCD_RW-WR: FSMC_NWE
LCD_A0: FSMC_A16

GPIO

BT_MFB: BM83 wake up
BT_RST_N: BM83 reset (active low)
BT_P0_0: uP wake up
GSEL0<1:0>
GSEL1<1:0>
GSEL2<1:0>
GSEL3<1:0>
EFFECT_SEL<3:0>
ADC_CS<3:0>
IRQMDAT<3:0>
LCD_E-RD: PE11 = ~CS1B
LCD_/RES
RENC_A [input]
RENC_B [input]
RENC_1 [input]
RENC_2 [input]
FOOTSWITCH_T1
FOOTSWITCH_T2

Should all be one time program events (can be atomic)

Interfaces by Functional Block

Pre-amp

GSEL{0,1,2,3}<1:0> (GPIO)
The value of GSELN<1:0> sets the rough gain at channel N:
00 = 0 dB (pass input)
01 = In-Amp gain
10 = In-Amp gain + 30dB
11 = Unused
POT_SCL & POT_SDA (I2C)
I2C bus for controlling the digital potentiometers. See the Digipot I2C Bus Address Table for addressing and the MCP4451 Datasheet for commands. Shared with the Analog Effect.

Analog Overdrive

EFFECT_SEL<3:0> (GPIO)
When EFFECT_SEL<N> is asserted, the Analog Effect is enabled on channel N.
POT_SCL, POT_SDA (I2C)
(shared with pre-amp)

ADC

MCLK (clock)
Master Clock supplied by the uP to the ADC(s). Should be at least 6 MHz to achieve desired sample rate.
ADC_SCK, ADC_MISO, ADC_MOSI (SPI)
SPI bus for the ADC(s) to receive commands from the microcontroller and report back data collected.
ADC_CS<3:0> (GPIO)
Active-low Chip Select lines for Channel 0-3 ADCs. Must assert to communicate with a particular ADC via SPI.
IRQMDAT<3:0> (GPIO)
Each ADC has an ~IRQ/MDAT pin, which can either be used to send interrupts to the microcontroller or output the divided clk frequency.

Bluetooth

BT_RFS, BT_SCLK, BT_DR (I2S)
BT_RXD, BT_TXD (UART)
BT_MFB (GPIO)
BM83 wakeup
BT_RST_N (GPIO)

BM83 reset

Reference:
BM83 design guide 3.0 AUDIO TRANSCEIVER SOLUTION (p. 28)
BM83 host MCU development guide 5. UART Communication Protocol (p. 29)

LCD

LCD_DB<7:0>, LCD_CS1B, LCD_RW-WR, LCD_A0 (FSMC)
LCD_E-RD (GPIO)
enable pin (6800), always ~CS1B
LCD_/RES (GPIO)
active low reset

Rotary Encoder (RE)

RENC_A, RENC_B (GPIO)
encoder channel A and B
RENC_1, RENC_2 (GPIO)
push button

Reference:
https://www.digikey.com/en/products/detail/bourns-inc/PEC12R-3220F-S0024/4699265

Foot Switch

FOOTSWITCH_T1, FOOTSWITCH_T2 (GPIO)
footswitch output throw 1 and 2

Reference:
https://www.amazon.com/Lovermusic-Plastic-Electric-Momentary-Non-latching/dp/B07CKB6PDV/ref=cm_cr_arp_d_pl_foot_top?ie=UTF8

Kernel Logic Flow

setup()

Main loop:

```
input = get_adc()        // get one unit of work from kernel adc read buffer
output = process(input)          // process one unit of work
ble_send(output)        // add processed work to i2s write buffer
```

Interrupts:
continuous adc read and I2S write updates

```
rotary_encoder_isr() {
        action = parse_re_data()
        update_lcd(action)
        If (select setting) {
                update system state variables stored in kernel   // take effect in next loop iteration
        }
}
lcd_isr(){ // continuous lcd updates  (only if RE is recently active)
        // isr called either by timer (periodic refresh), or by some sort of TxE
        write kernel buffered updated display to the data bus
}
```

All other continuous control signals

Function details:

byte[] process(byte[] input)
Process the audio piece based on digital effect enable flags
If no effects, output = input

update_lcd(action)
Compute new display data from RE action and update kernel data structure
* this function should NOT block
Then call lcd_isr or wait for periodic refresh

Digital Signal Processing