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

LCD_DB6: SPI1_SCK
LCD_DB7: SPI1_MOSI

DMA2 stream 3 channel 3, SPI1_TX

SPI / TDM

ADC_BCK: I2S3_CK
ADC_LRCK: I2S3_WS
ADC_DOUT: I2S3_SD

DMA1 stream 0 channel 0, SPI3_RX

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
Bus: APB1

DMA1 stream 4 channel 0, SPI2_TX

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~~
~~Bus: AHB3~~

GPIO

BT_MFB: BM83 wake up
BT_RST_N: BM83 reset (active low)
BT_P0_0: uP wake up
CH<3:0>_SDB: pre-amp channel shutdown
EFFECT_SEL<3:0>
ADC_CS<3:0>
ADC_IRQ
LCD_RW-WR
LCD_A0
LCD_CS1B, LCD_E-RD = ~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

CH<3:0>_SDB (GPIO)
Active-low, assert CHN_SDB to shut down the amplifier for channel N.

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_BCK, ADC_LRCK, ADC_DOUT (SPI / TDM)
SPI bus for 4-channel TDM data from ADC
ADC_IRQ (GPIO)
Clip detection
SCL, SDA (I2C)
Control; shared with pre-amp and analog effect

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)

Setting BM83 parameters: (e.g. I2S slave mode)

1. Creating .HEX file: IS2083/BM83 Bluetooth Applications Design Guide Appendix B (p. 46)
2. Programming the .HEX file: BM83 Bluetooth Audio Development Board User's Guide 5. Firmware Update
   "It is possible to update only this config file by only selecting this .HEX file in the update process and selecting image number to 1 in the isUpdate tool."

LCD

LCD_DB6, LCD_DB7 (SPI)
SPI slave with only input data (MOSI)
LCD_RW-WR (GPIO)
read/write select (6800) (0=write; 1=read)
LCD_A0 (GPIO)
register select (0=command; 1=data)
LCD_CS1B, LCD_E-RD (GPIO)
E-RD is enable pin (6800), always ~CS1B
LCD_/RES (GPIO)
active low reset

Reference:
https://www.digikey.com/en/products/detail/newhaven-display-intl/NHD-C12864WC-FSW-FBW-3V3-M/2626409

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 SPI read buffer; properly deal with channel TDM by seperating data for each channel
output = process(input) // process one unit of work
ble_send(output) // add processed work to BT I2S write buffer

Interrupts:
DMA handles continuous ADC SPI read and BT I2S write updates
DMA handles LCD SPI writes

```
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
        }
}
```

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 DMA will gradually send out all the updated data through SPI

<u>Other features To-do:</u>
Clipping indicator
Automatic gain control (ADC clip -> ADC auto control AND/OR turn down pre-amp gain) with enable/disable

Latency Calculations

Data rate: 88.2 KB/s per channel
Put kernel SRAM data size upper bound into memory calc sheet!

SRAM vs. Flash
Flash writes:
“the internal flash memory controller in the STM32's won't allow any writes unless the entire page is cleared.”
(https://electronics.stackexchange.com/questions/433401/how-to-properly-use-stm32-flash-memory-as-an-eeprom)

Data sheet:
RAM memory is accessed (read/write) at CPU clock speed with 0 wait states
Flash: 168 MHz with 5 wait states; no wait state w/ ART (only for read)
Table 40. Flash memory programming gives erase time for 16K (400ms) to 128K (2s) sectors
Memory we need (especially for reverb alone): see analysis

SPI + DMA latency & bandwidth
“To operate at its maximum speed, the SPI needs to be fed with the data for transmission and the data received on the Rx buffer should be read to avoid overrun. To facilitate the transfers, the SPI features a DMA capability” (reference manual 28.3.9)

1MB/s baud rate - should be sufficient

Digital Signal Processing