10555 Scripps Trl, San Diego, CA 92131
858-877-3485 858-408-9550
http://www.pulserain.com

Doc# TRM-0922-01001, Rev 1.0.1

PulseRain M10 – Voice CODEC

Technical Reference Manual

Aug, 2017

This page is intentionally left blank.

Table of Contents

References

1. Si3000 Voice Band CODEC with Microphone / Speaker Drive, Rev 1.4, Silicon Laboratories, 12/2010

1 Introduction

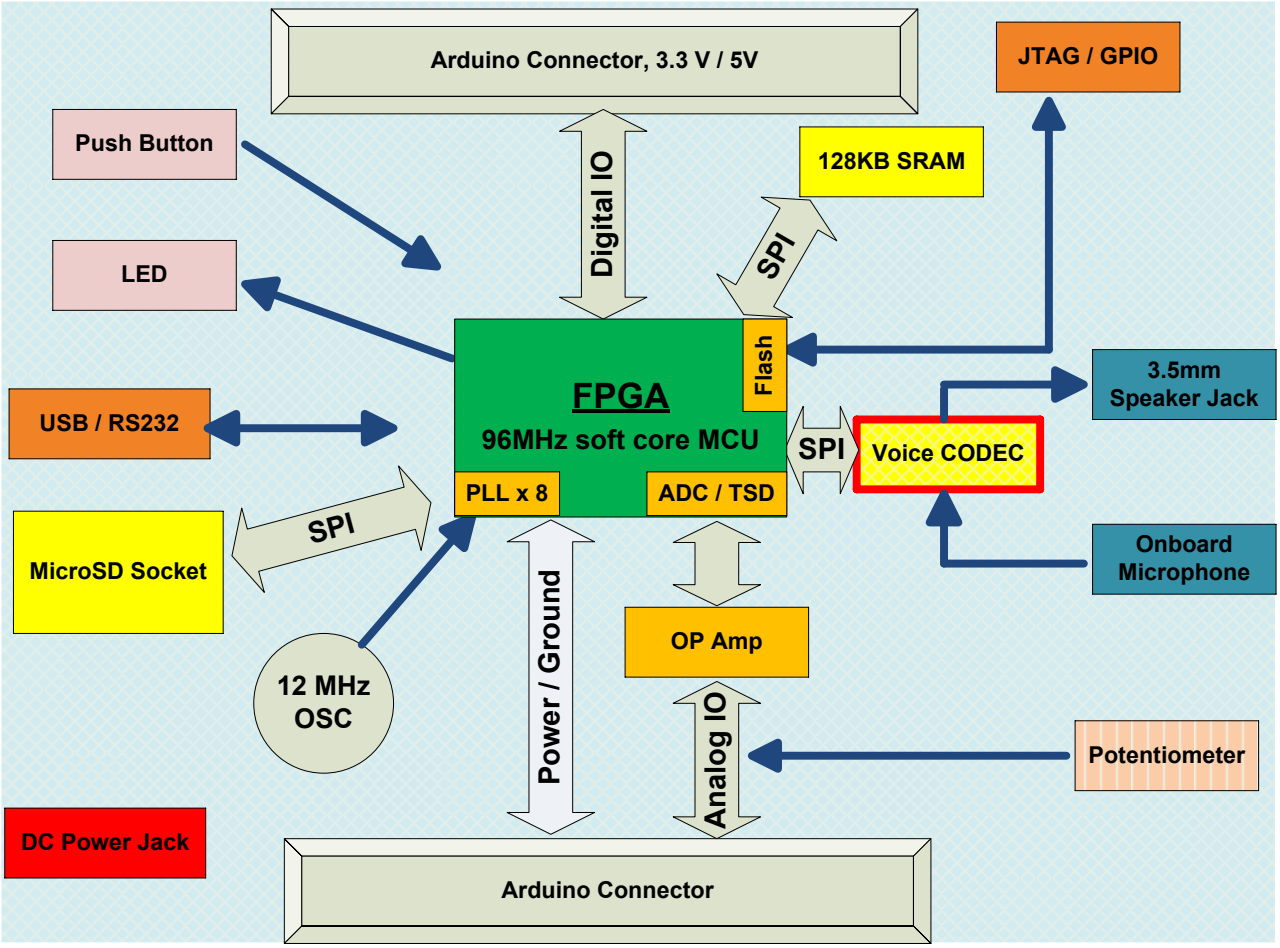

Figure 1-1 The Close View of M10

As shown Figure 1-1, the M10 board takes a distinctive technical approach by embedding an open source soft MCU core (96MHz) into an Intel MAX10 FPGA, while offering an Arduino compatible software interface and form factors. Among all the onboard peripherals, there is a voice CODEC to interface with the speaker and microphone. Accordingly, PulseRain Technology has designed an open source controller to interact with the CODEC from the FPGA side. And this document contains the technical detail of this controller.

2 Hardware

2.1 Architecture

The part number for the onboard CODEC chip is **Si3000** from Silicon Labs. It has 4 different serial modes (Ref [1]), and only **mode 0** is supported by M10.

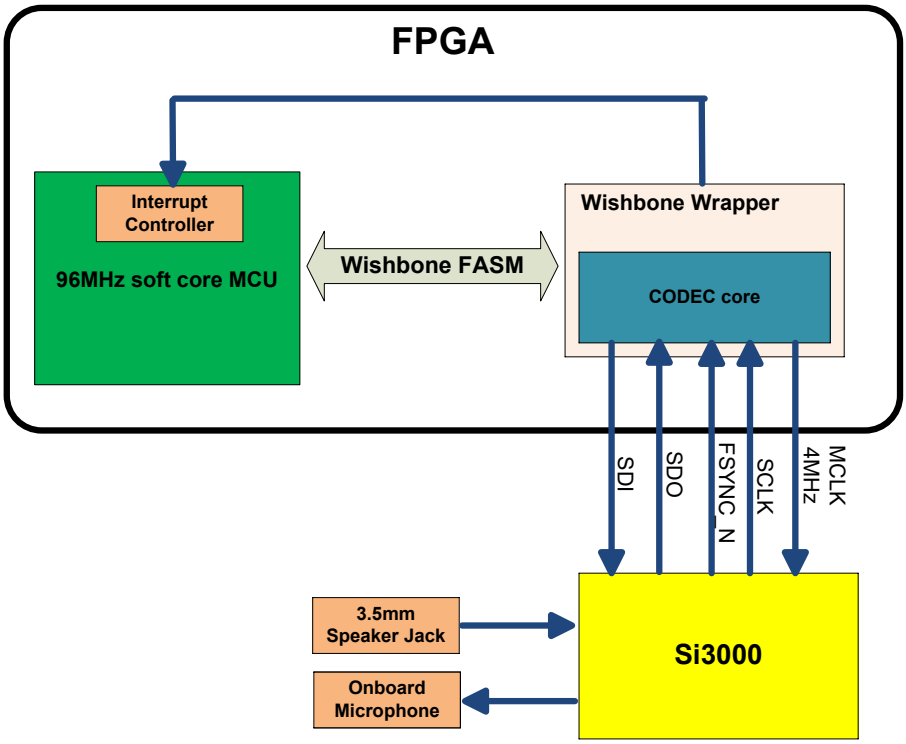

Figure 2-1 The FPGA Controller for Si3000

As shown in Figure 2-1, the FPGA controller for Si3000 is mainly composed of two parts: the CODEC core and the Wishbone wrapper. The wishbone wrapper is used to communicate with the MCU core for register read/write. And the CODEC core communicates with the Si3000 through 5 pins:

- MCLK: the master clock (4MHz), from FPGA to Si3000
- SCLK: the slave clock (2048KHz), from Si3000 to FPGA. The SCLK runs at 256 bits per frame. For M10 board, the **default sample rate is set to be 8KHz**, so 8KHZ * 256 = 2048KHz
- FSYNC_N: Frame Sync, level sensitive signal that goes all the way low during the active cycles
- SDO: Serial Data from Si3000 to FPGA
- SDI: Serial Data from FPGA to Si3000

Interrupt is also supported by the FPGA controller. A pulse triggered interrupt will be generated when a valid sample is received or a new sample has been completely transmitted.

2.2 Port List

The port list of the CODEC core is defined in Table 2-1.

Group Name	Signal Name	In/Out	Bit Width	Description
Clock / Reset	clk	Input	1	Clock input, 96MHz
	reset	Input	1	Asynchronous reset, active low
	sync_reset	Input	1	Synchronous reset, active high
	mclk_enble	Input	1	Enable master clock
Host Interface	write_data	Input	16	Voice sample to be transmitted from FPGA to Si3000
	write_data_grasp	Output	1	A pulse to indicate that the write_data has been transmitted
	read_data	Output	16	Voice sample received from Si3000
	done	Output	1	A pulse to indicate that a Si3000 (read/write) frame has been processed completely.
Si3000 Interface	Si3000_SDO	Input	1	Serial Data from Si3000 to FPGA
	Si3000_SDI	Output	1	Serial Data from FPGA to Si3000
	Si3000_SCLK	Input	1	Slave clock from Si3000 to FPGA
	Si3000_FSYNC_N	Input	1	Frame (active low) signal from Si3000 to FPGA
	Si3000_MCLK	Output	1	Master clock from FPGA to Si3000
	fsync_out	Output	1	A delayed version (CDC processed) of Si3000_FSYNC_N

Table 2-1 Port List of CODEC Core

2.3 Repository

The RTL code for CODEC core and its Wishbone wrapper is part of PulseRain Technology's RTL library, which can be found on GitHub:

https://github.com/PulseRain/PulseRain_rtl_lib

3 Software

3.1 Register Definition

The Wishbone wrapper shown in Figure 2-1 contains all the registers to control the CODEC core. In a nutshell, the registers are defined as following:

- WRITE_DATA_LOW (8 bit) and WRITE_DATA_HIGH (8 bit)
  These two write-only registers contain the lower 8 bits and higher 8 bits for sample write. For voice output (FPGA to Si3000, then to the Speaker), the output sample (16 bit) should be written to those two registers before start the frame transmission. However, out of the 16 bits, only the higher 15 bits represent the actual voice data. The LSB is replaced by a control bit in order to work with Si3000.

- READ_DATA_LOW (8 bit) and READ_DATA_HIGH (8 bit)
  For voice input (Microphone to Si3000, then to the FPGA), the input sample (16-bit sample) is save in those two registers. And they are read-only

- CSR (Control Status Register)
  The bits for CSR are defined in Table 3-1:

Bits	R/W	Default	Description
0	WO	0	Write 1 to Sync Reset. Always return 0 for read
1	RW	0	Write 1 to Enable the CODEC. (The MCLK will be off is CODEC is disabled)
2	RO	0	Write busy flag. This bit will be set when data is written to WRITE_DATA_LOW register. And this bit will be cleared after the sample is sent to Si3000. So please observe this flag before write new samples.
3	RO	0	Data available flag. This bit will be set after new sample is received from Si3000. And this bit will be cleared after READ_DATA_LOW register is being read.
4	RO	0	This bit will go low when FSYNC_N is asserted (from high to low). And this bit will go high when a full frame of data (16 bits) has been received.
7:5	RO	0	RESERVED, always read as zero

Table 3-1 Bit Map for CSR (Control Status Register)

3.2 Address Map

The registers defined in Section 3.1 are mapped into MCU's address space, as shown in Table 3-2.

Address	Register Name
0xE8	CODEC_WRITE_DATA_LOW
0xE9	CODEC_WRITE_DATA_HIGH
0xEA	CODEC_READ_DATA_LOW
0xEB	CODEC_READ_DATA_HIGH
0xEC	CODEC_CSR

Table 3-2 Address Definition

3.3 Work Flow

As mentioned in Ref [1], Si3000 has two kinds of frames, with which the primary frame is for digital audio data samples, and secondary frame is for accessing internal registers. And those two kinds of frames are distinguished by the LSB of the 16-bit data. Secondary frame is requested only when the LSB of the previous primary frame is 1, and the secondary frame occurs between primary frames.

The FPGA controller for Si3000 has consolidated the two kinds of frames into general register read/write.

- To write 8-bit data into an internal register, do the following:
  1. wait for the write busy flag (bit 2) in CODEC_CSR to be de-asserted.
  2. write the address of internal register into CODEC_WRITE_DATA_HIGH
  3. write 1 to CODEC_WRITE_DATA_LOW
  4. wait for the write busy flag (bit 2) in CODEC_CSR to be de-asserted.
  5. write the 8-bit data into CODEC_WRITE_DATA_LOW
  6. wait for the write busy flag (bit 2) in CODEC_CSR to be de-asserted.
  7. write 0 into CODEC_WRITE_DATA_HIGH
  8. write 0 into CODEC_WRITE_DATA_LOW
  9. wait for the write busy flag (bit 2) in CODEC_CSR to be de-asserted.

- To read 8-bit data from an internal register, do the following:
  1. read CODEC_READ_DATA_LOW
  2. write the 5 bit address of the internal register into CODEC_WRITE_DATA_HIGH[4:0]
  3. set CODEC_WRITE_DATA_HIGH[5] to be 1
  4. wait for the data available flag (bit 3) in CODEC_CSR to be asserted
  5. wait for the write busy flag (bit 2) in CODEC_CSR to be de-asserted
  6. write 1 to CODEC_WRITE_DATA_LOW
  7. read CODEC_READ_DATA_LOW
  8. wait for the write busy flag (bit 2) in CODEC_CSR to be de-asserted
  9. write 0 to CODEC_WRITE_DATA_LOW
  10. wait for the data available flag (bit 3) in CODEC_CSR to be asserted
  11. read CODEC_READ_DATA_LOW
  12. wait for the data available flag (bit 3) in CODEC_CSR to be asserted
  13. read CODEC_READ_DATA_LOW to get the register value

- To write 16-bit audio sample to the Si3000, do the following (The LSB of the 16-bit sample will be set to zero as required by Si3000. Please see Ref[1] for more detail.):
  1. wait for the write busy flag (bit 2) in CODEC_CSR to be de-asserted
  2. write the higher 8 bits of the audio sample into CODEC_WRITE_DATA_HIGH
  3. write the lower 8 bits of the audio sample into CODEC_WRITE_DATA_LOW, with LSB being set to zero

- To read 16-bit audio sample from the Si3000, do the following:
  1. wait for the data available flag (bit 3) in CODEC_CSR to be asserted
  2. read CODEC_READ_DATA_HIGH for the higher 8 bits of the audio sample
  3. read CODEC_READ_DATA_LOW for the lower 8 bits of the audio sample

3.4 Arduino Library

3.4.1 APIs

To turn the workflows in Section 3.3 into API calls, PulseRain Technology has come up with the M10CODEC library, which covers the following:

- *void begin()*

  call this function in the beginning to initialized the Si3000

- *uint16_t regRead (uint8_t addr)*

  call this function to read the internal registers of Si3000

- *void regWrite (uint8_t addr, uint16_t data)*

  call this function to write the internal registers of Si3000

- *uint16_t sampleRead ()*

  call this function to read a 16-bit audio sample from Si3000

- *void sampleWrite (uint16_t data)*

  call this function to write a 16-bit audio sample to Si3000

- *uint8_t sampleCompress(int16_t pcm_val)*

  call this function to compress 16-bit audio sample into 8-bit data according to μ Law

- *int16_t sampleExpand (uint8_t u_val)*

  call this function to restore an 8-bit data to 16-bit audio sample according to μ Law

- *void outputVolume(uint8_t volume)*

  call this function to adjust the output volume. The valid value of volume is from 0 to 32. A value of 0 will mute the output, and a value of 32 will set the TX PGA gain of Si3000 to be 12dB. For every step of volume value, a delta of 1.5dB will be applied to TX PGA gain.

3.4.2 Examples

To further facilitate the software development, the M10CODEC library has also provided a few examples:

- *wav_play*
  This example will read a wav file named "SPIDER.WAV" from the microSD card, and play it through Si3000. This wav file is supposed to be mono channel, sampled at 8KHz.

  To run this example, find a microSD card and format it with FAT32. The M10CODEC library contains the "SPIDER.WAV" file in the "extras" folder. Copy this file to the root of the microSD card. And then run the example of wav_play.ino in Arduino IDE to hear the sound playing. (A speaker or headset has to be connected to the M10 board through 3.5mm audio jack).

- *wav_record_play*
  This example will record 16 seconds of sound into M10's onboard SRAM, and play it back later. (To save memory, the audio samples are compressed with μ Law.

  Before running the example, install Python 3 (https://www.python.org) and make it available through the environment variable of PATH. Python 3 is needed to run the scripts on the host side along with the firmware.

  The sound is recorded through onboard microphone. To play it back, a speaker or headset has to be connected to the M10 board through 3.5mm audio jack.

  To run the example, run the wav_record_play.ino in Arduino IDE. And then open a command prompt and enter the "extras" folder of the M10CODEC library. Type in "Python wav_console.py COMx" to start the wave console, for which the COMx is the COM port connected to the M10 board.

  wav_console.py is a command line program. Type in "help" to get the available commands. use "record" command to record sound up to 16 seconds. Use "play" to play the recorded sound repetitively. (Press enter to stop the playing).

3.4.3 Interrupt

As illustrated in Figure 2-1, the FPGA controller supports interrupt. Accordingly, the software will have a way to setup the ISR. This done by using the function *attachIsrHander()* with an IRQ index of 6, like the following:

```
#define CODEC_INT_INDEX 6

void codec_isr_handler() { ... } // The ISR function

attachIsrHandler(CODEC_INT_INDEX, codec_isr_handler); // setup the ISR
```

And the example of *wav_play* in Section 3.4.2 uses ISR to send samples to Si3000.