STM32-Nucleo Shield for real-time audio acquisition and processing

Project made by Daidone Luca and Dedor Alessio as part of “Advanced Operating Systems” and “Embedded Systems” courses

Academic year 2023/2024

POLITECNICO
MILANO 1863

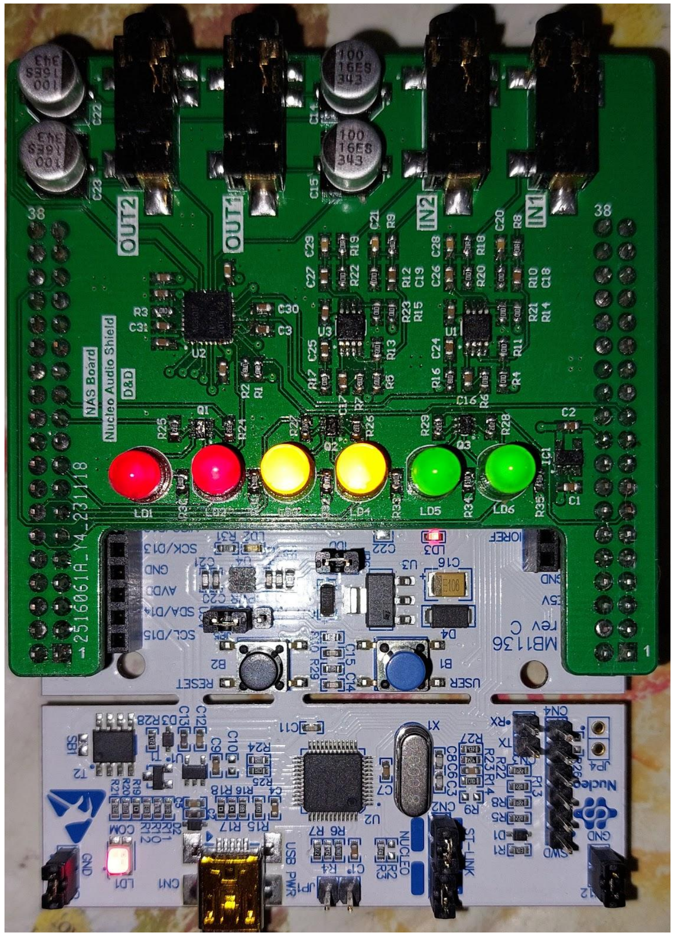
INTRODUCTION

Project goals

Design of a **TLV320AIC3101 Codec** based shield targeting **STM32F401RE Nucleo** boards and development of a **multi-threaded** program for **real-time** audio data processing.

Aim of the project is to:

1. Design and realize a custom “shield” to add audio processing capabilities to the STM32 Nucleo boards
2. Use the embedded RTOS “Miosix” to implement an I2S Driver and exploit the advantages of multi-threaded programming



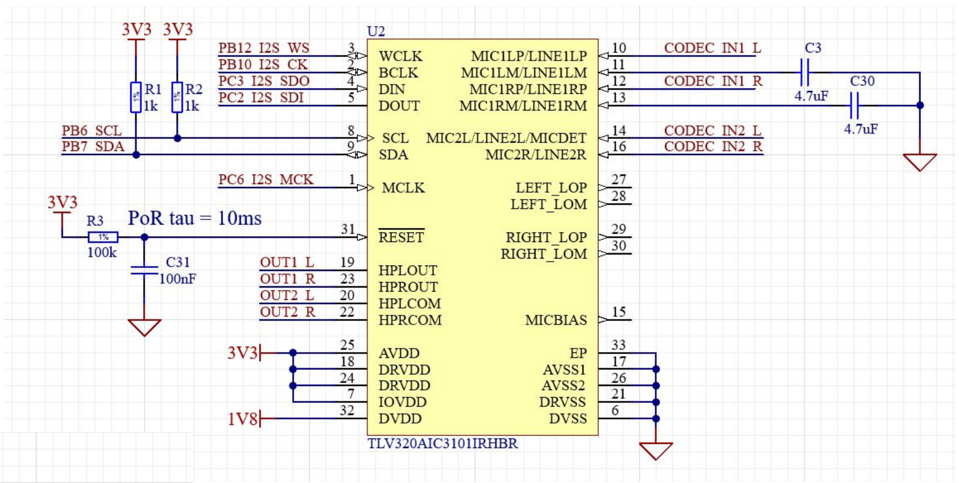
POLITECNICO
MILANO 1863

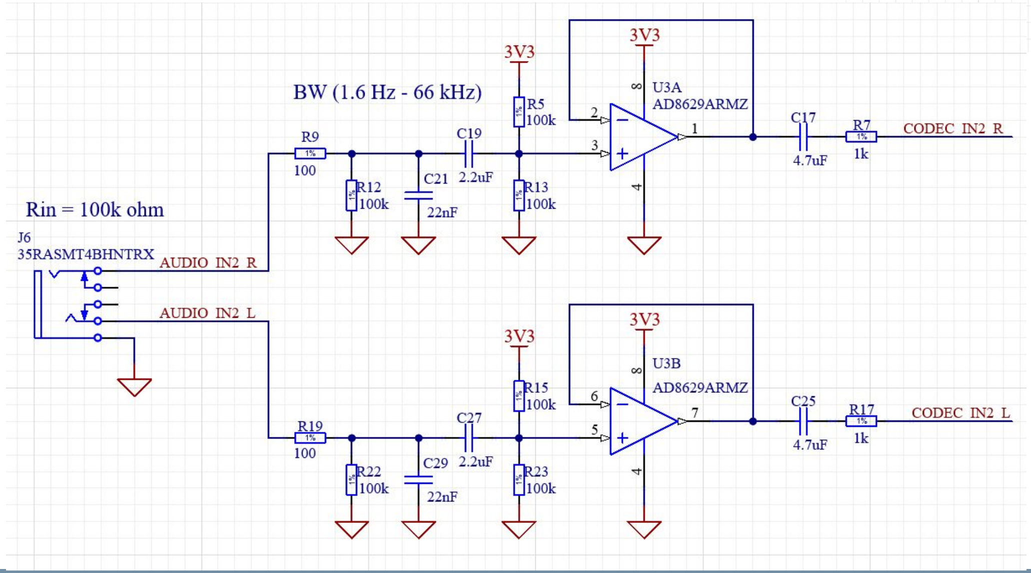
HARDWARE

Starting from the schematic..

First step is the design of the schematic which includes:

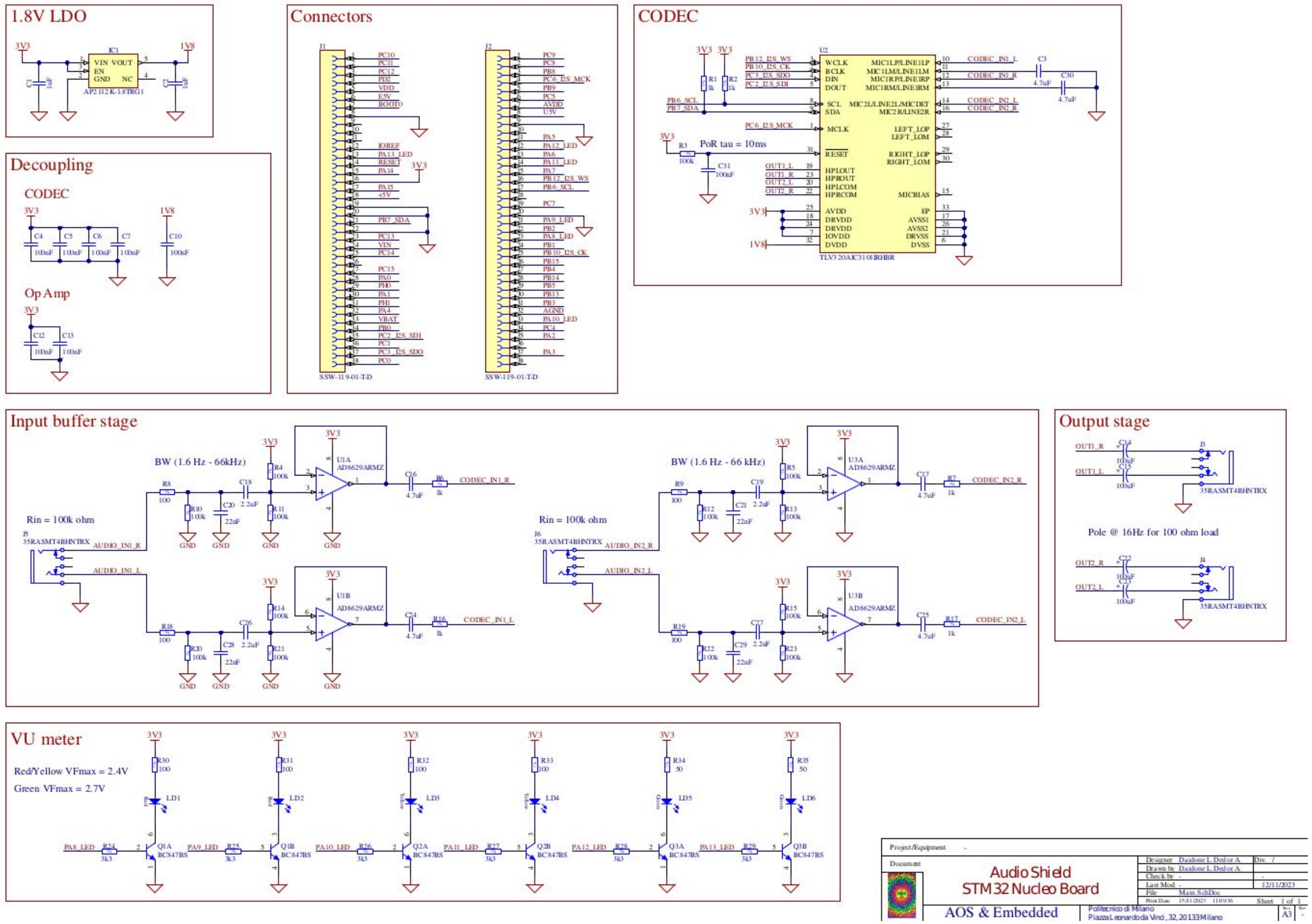
- Analog front end
- LDO
- Codec
- VU-meter





POLITECNICO MILANO 1863

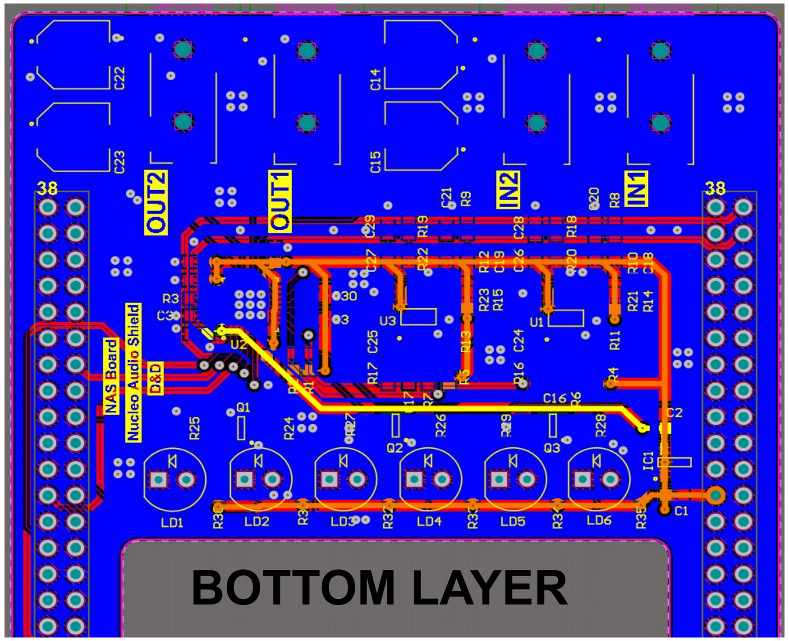
Full schematic

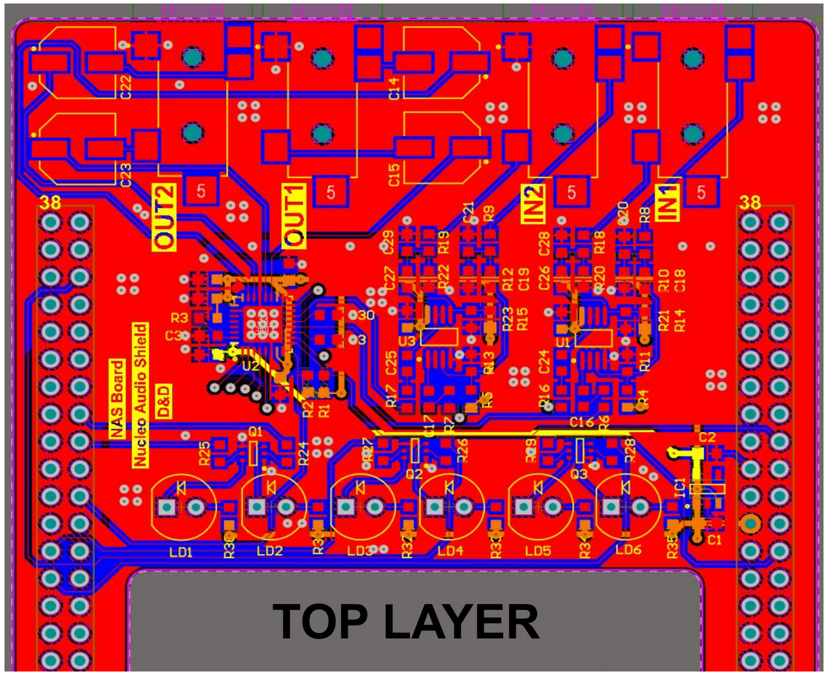


..continuing to the Layout

Then we move to the **PCB design**:

- 2 layers board designed with Altium Designer and produced by JLCPCB





..and finishing with soldering + testing

Assembly and first tests…

Final considerations (HW)

Problems encountered:

1. **VU-meter** was meant to have 6 fully working LEDs but only 5 of them can be currently used. One is directly connected to the SWD port and setting it as an output would not allow us to successfully “flash” the MCU using the “st-flash” tool. This can probably be easily fixed but was not investigated yet

2. **Signal Integrity:** due to initial misunderstanding about the I2S protocol, a trace carrying audio signals was routed close to traces carrying high frequency digital signals causing some audible distortion under certain conditions

3. The selected **Codec** comes in a package which is not easy to solder. A too long heat exposition damaged the output audio stage

POLITECNICO
MILANO 1863

SOFTWARE

Goals

Software must provide libraries/functions to easily manage all the HW present on the board, which are:

- Codec
- VU-meter
- I2S peripherals on codec and STM32

Finally, a simple loopback and a VU-meter showcase should be realized.

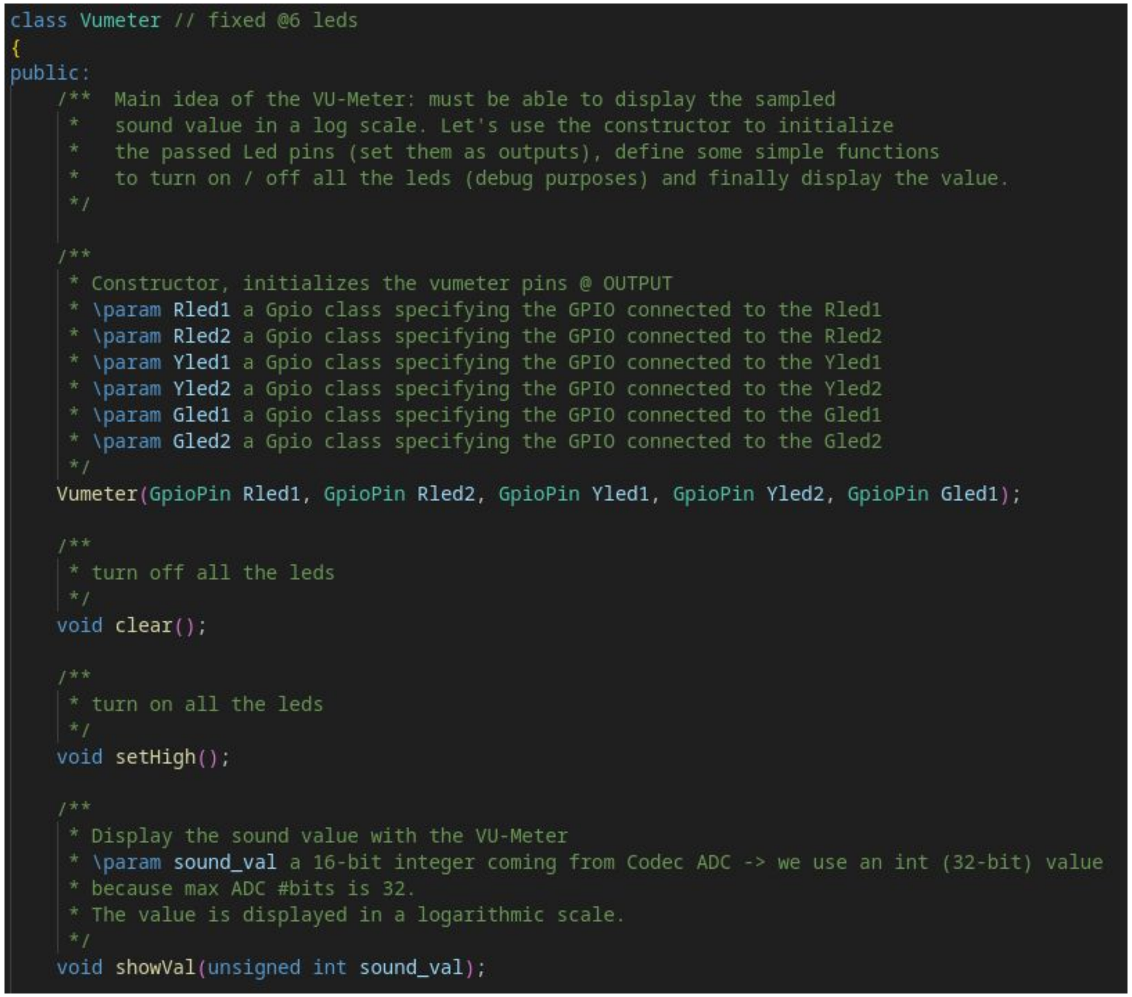
VU-Meter

The class "**Vumeter**" takes the specified GpioPin (5 pins) and provides some basic **methods.**

- The **constructor** set the specified pins, passed as parameters, as "outputs".
- *setHigh()* and *clear()* allow to turn on/off all the leds of the VU-meter.
- *showVal(uint)*, finally takes the sound value, converts it in a log scale and turns on the corresponding number of leds.

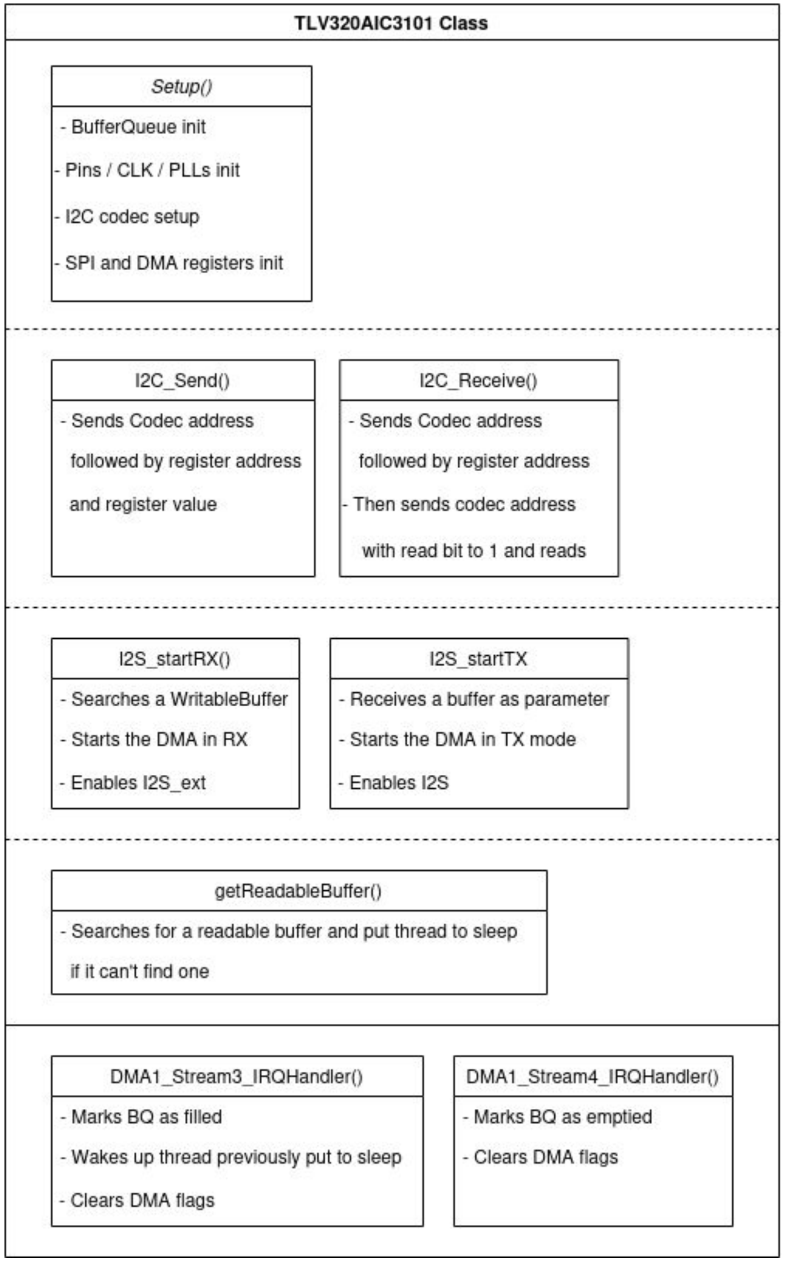
The class is defined inside the vumeter.h header file.

VU-Meter



```
class Vumeter // fixed @6 leds
{
public:
    /**  Main idea of the VU-Meter: must be able to display the sampled
     *   sound value in a log scale. Let's use the constructor to initialize
     *   the passed Led pins (set them as outputs), define some simple functions
     *   to turn on / off all the leds (debug purposes) and finally display the value.
     */

    /**
     * Constructor, initializes the vumeter pins @ OUTPUT
     * \param Rled1 a Gpio class specifying the GPIO connected to the Rled1
     * \param Rled2 a Gpio class specifying the GPIO connected to the Rled2
     * \param Yled1 a Gpio class specifying the GPIO connected to the Yled1
     * \param Yled2 a Gpio class specifying the GPIO connected to the Yled2
     * \param Gled1 a Gpio class specifying the GPIO connected to the Gled1
     * \param Gled2 a Gpio class specifying the GPIO connected to the Gled2
     */
    Vumeter(GpioPin Rled1, GpioPin Rled2, GpioPin Yled1, GpioPin Yled2, GpioPin Gled1);

    /**
     * turn off all the leds
     */
    void clear();

    /**
     * turn on all the leds
     */
    void setHigh();

    /**
     * Display the sound value with the VU-Meter
     * \param sound_val a 16-bit integer coming from Codec ADC -> we use an int (32-bit) value
     * because max ADC #bits is 32.
     * The value is displayed in a logarithmic scale.
     */
    void showVal(unsigned int sound_val);
```

Codec driver: TLV320AIC3101 class


TLV320AIC3101 Class
Setup()
- BufferQueue init
- Pins / CLK / PLLs init
- I2C codec setup
- SPI and DMA registers init
I2C_Send()
- Sends Codec address followed by register address and register value
I2C_Receive()
- Sends Codec address followed by register address
- Then sends codec address with read bit to 1 and reads
I2S_startRX()
- Searches a WritableBuffer
- Starts the DMA in RX
- Enables I2S_ext
I2S_startTX
- Receives a buffer as parameter
- Starts the DMA in TX mode
- Enables I2S
getReadableBuffer()
- Searches for a readable buffer and put thread to sleep if it can't find one
DMA1_Stream3_IRQHandler()
- Marks BQ as filled
- Wakes up thread previously put to sleep
- Clears DMA flags
DMA1_Stream4_IRQHandler()
- Marks BQ as emptied
- Clears DMA flags

Codec

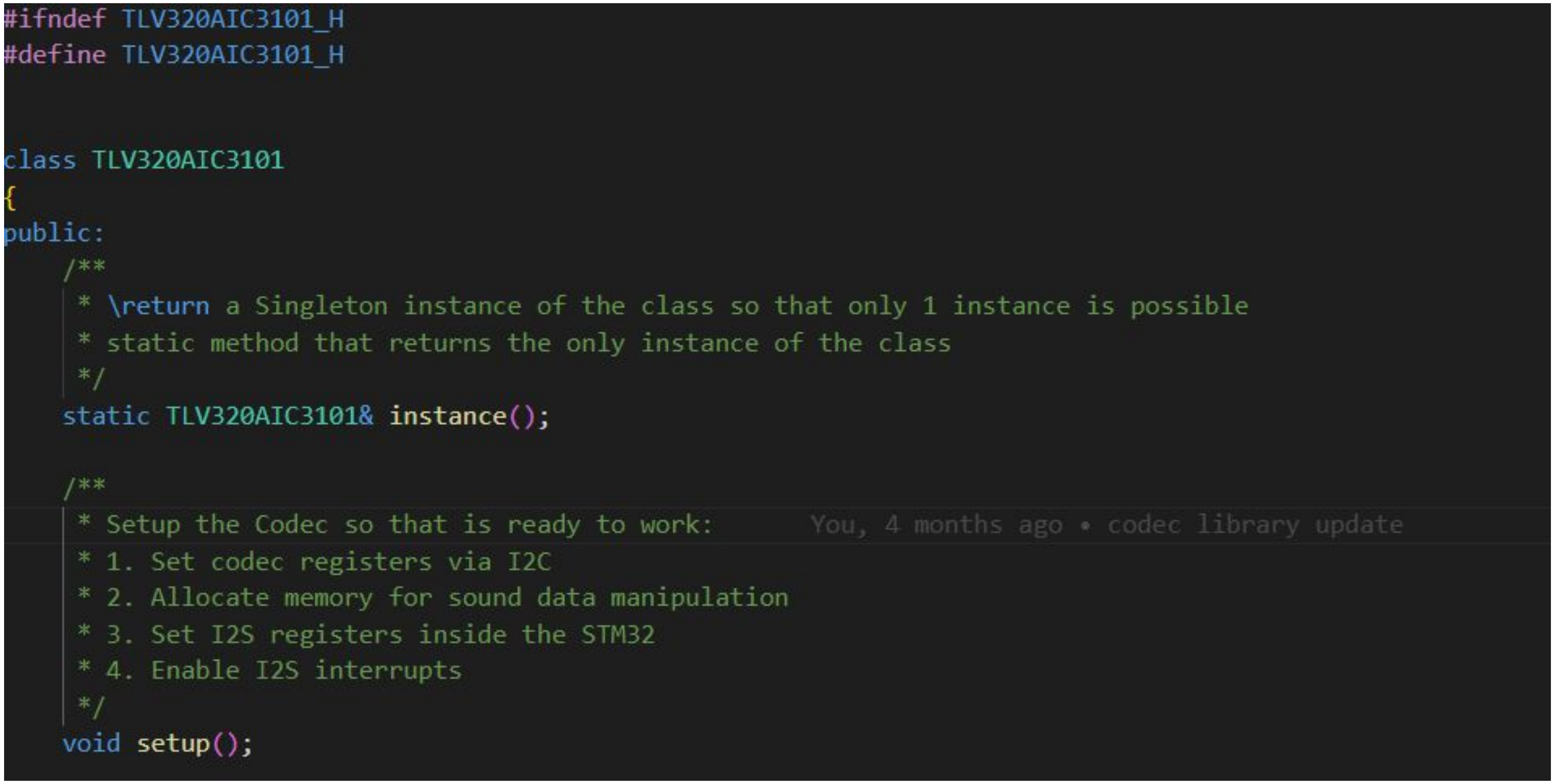
The class “**TLV3203101AIC**” instantiate the codec object in a Singleton manner such as only one instance can exist.

The codec uses two different communication protocols:

- **I2C**, it is needed to change the codec settings/functionalities and directly R/W its registers
- **I2S**, used to receive/transfer audio data

The class is defined inside the TLV3203101AIC.h header file, while all the implementations are moved inside the TLV3203101AIC.cpp file.

Codec

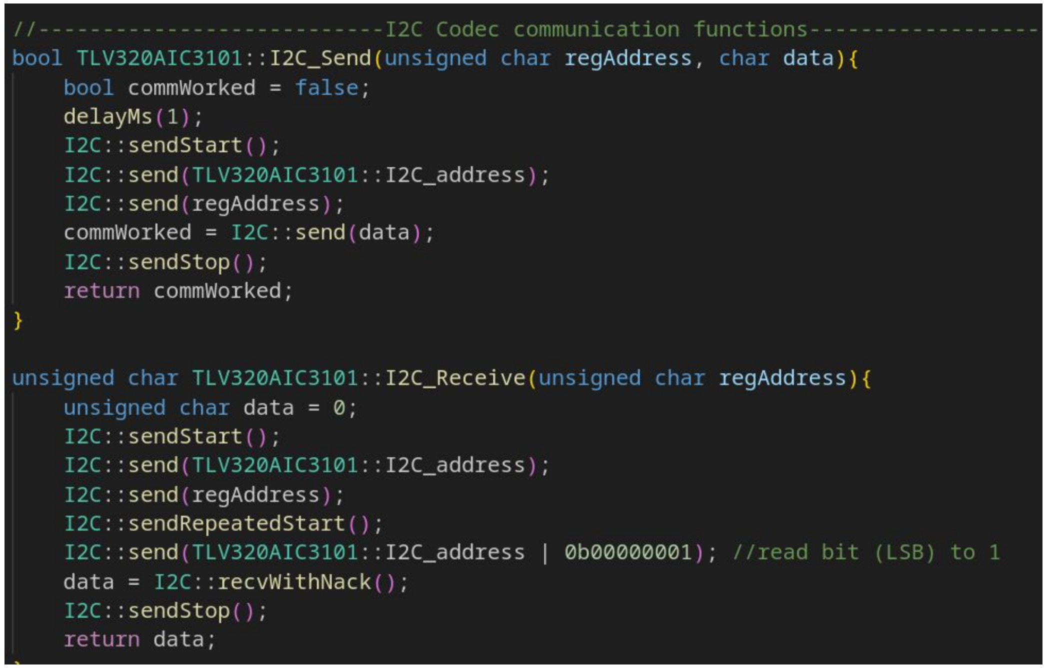


```
#ifndef TLV320AIC3101_H
#define TLV320AIC3101_H


class TLV320AIC3101
{
public:
    /**
     * \return a Singleton instance of the class so that only 1 instance is possible
     * static method that returns the only instance of the class
     */
    static TLV320AIC3101& instance();

    /**
     * Setup the Codec so that is ready to work:      You, 4 months ago • codec library update
     * 1. Set codec registers via I2C
     * 2. Allocate memory for sound data manipulation
     * 3. Set I2S registers inside the STM32
     * 4. Enable I2S interrupts
     */
    void setup();
```

Codec - I2C

In order to easily communicate through I2C protocol:

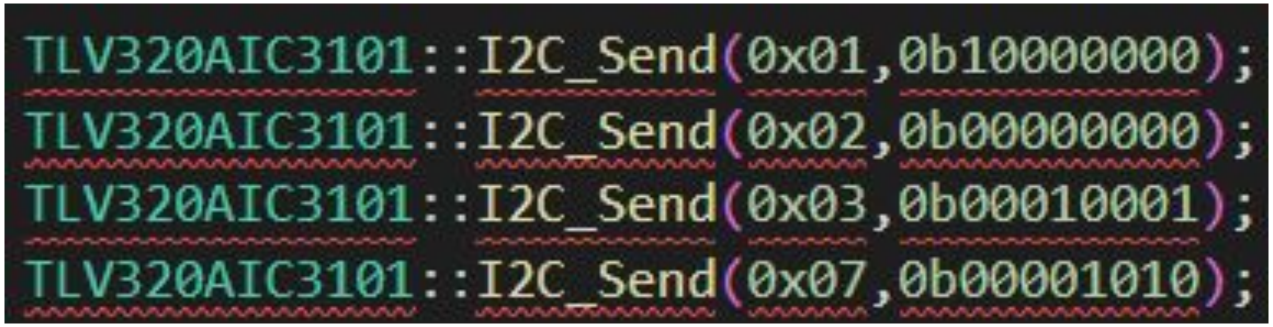
1. We used the “SoftwareI2C” class of Miosix to implement two methods, “`I2C_Send(unsigned char regAddress, char data)`“ and “`I2C_Receive(unsigned char regAddress)`“



```
//---------------------------I2C Codec communication functions-----------------
bool TLV320AIC3101::I2C_Send(unsigned char regAddress, char data){
    bool commWorked = false;
    delayMs(1);
    I2C::sendStart();
    I2C::send(TLV320AIC3101::I2C_address);
    I2C::send(regAddress);
    commWorked = I2C::send(data);
    I2C::sendStop();
    return commWorked;
}

unsigned char TLV320AIC3101::I2C_Receive(unsigned char regAddress){
    unsigned char data = 0;
    I2C::sendStart();
    I2C::send(TLV320AIC3101::I2C_address);
    I2C::send(regAddress);
    I2C::sendRepeatedStart();
    I2C::send(TLV320AIC3101::I2C_address | 0b00000001); //read bit (LSB) to 1
    data = I2C::recvWithNack();
    I2C::sendStop();
    return data;
```

Codec - I2C

2. Then we implemented a “setup()” method where the “I2C_Send()” function is used to setup the I2C peripheral and initialize all the needed codec registers.



```
TLV320AIC3101::I2C_Send(0x01,0b10000000);
TLV320AIC3101::I2C_Send(0x02,0b00000000);
TLV320AIC3101::I2C_Send(0x03,0b00010001);
TLV320AIC3101::I2C_Send(0x07,0b00001010);
```

And so on…

Codec - I2S

Audio data is sampled by the codec and then transferred to the MCU through I2S protocol. Since our goal is to receive and then send back the manipulated audio data, we need to exploit the I2S in **full-duplex** mode. Also, in order to avoid keeping the CPU busy, we use the **DMA** and its interrupt routines.

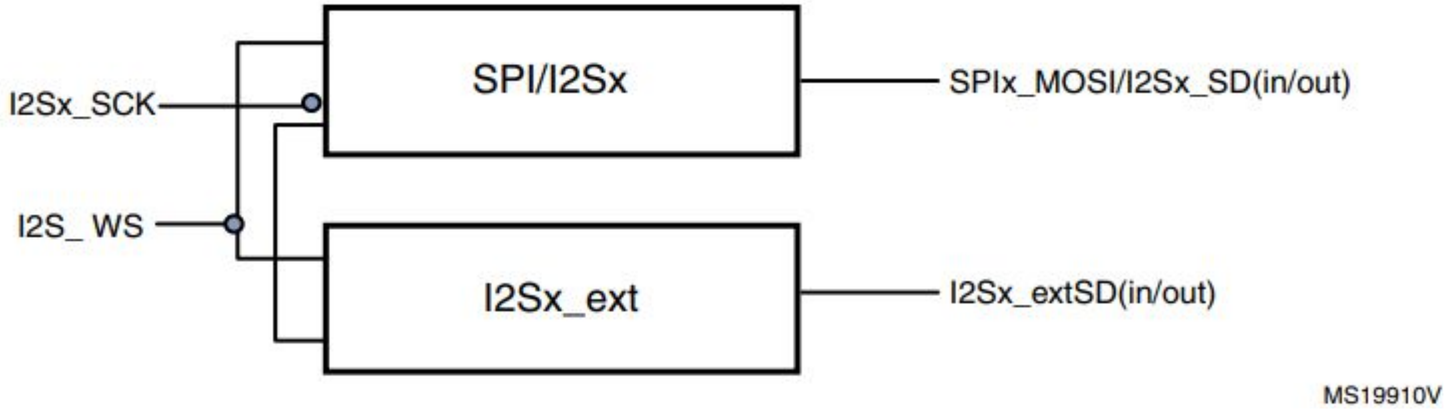
Therefore an “**I2S driver**” is needed. For this purpose, the TLV320AIC3101 class has the following methods:

- `void setup();`
- `static bool I2S_startRx();`
- `static void I2S_startTx(const unsigned short *buffer_tx);`

Codec - I2S - setup()

In the "**setup()**" function the following steps are performed:

- enable peripherals clock (DMA1/SPI2)
- setup I2S pins (with correct AF number)
- configure I2S PLL
- configure I2S2/I2S2_ext working mode (Master/Slave)
- setup relative DMAx streams
- enable interrupts



Codec - I2S - I2S_startRx()/I2S_startTx()

The role of the “**I2S_startRx()**” function is to:

- find a writable buffer among the ones of the BufferQueue
- Set DMA1 (Stream 3) to transfer from peripheral (I2S) to memory, then start it
- enable I2S2_ext peripheral

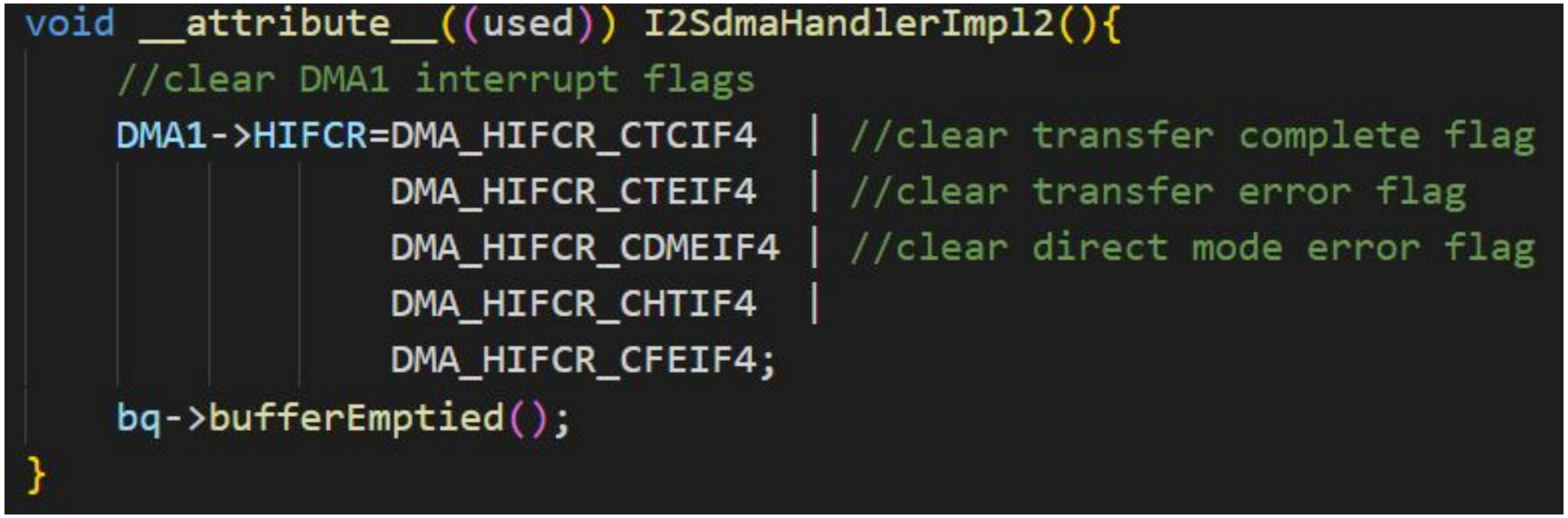
Similarly the role of the “**I2S_startTx()**” function is to:

- take the buffer filled with audio data as a parameter
- Set DMA1 (Stream 4) to transfer from memory to peripheral, then start it
- enable SPI2 (I2S2) peripheral

Note that both these methods use a “`FastInterruptDisableLock dLock;`” before setting everything up.

DMA1 ISR

Since both DMA1 Stream3/Stream4 interrupts have been enabled, we need to manage the relative **Interrupt Service Routines**. Apart from clearing all the interrupts flags we basically just mark the buffer which was just transmitted as “empty” and the one that was received as “filled”.



```
void __attribute__((used)) I2SdmaHandlerImpl2(){
    //clear DMA1 interrupt flags
    DMA1->HIFCR=DMA_HIFCR_CTCIF4  | //clear transfer complete flag
                DMA_HIFCR_CTEIF4  | //clear transfer error flag
                DMA_HIFCR_CDMEIF4 | //clear direct mode error flag
                DMA_HIFCR_CHTIF4  |
                DMA_HIFCR_CFEIF4;
    bq->bufferEmptied();
}
```

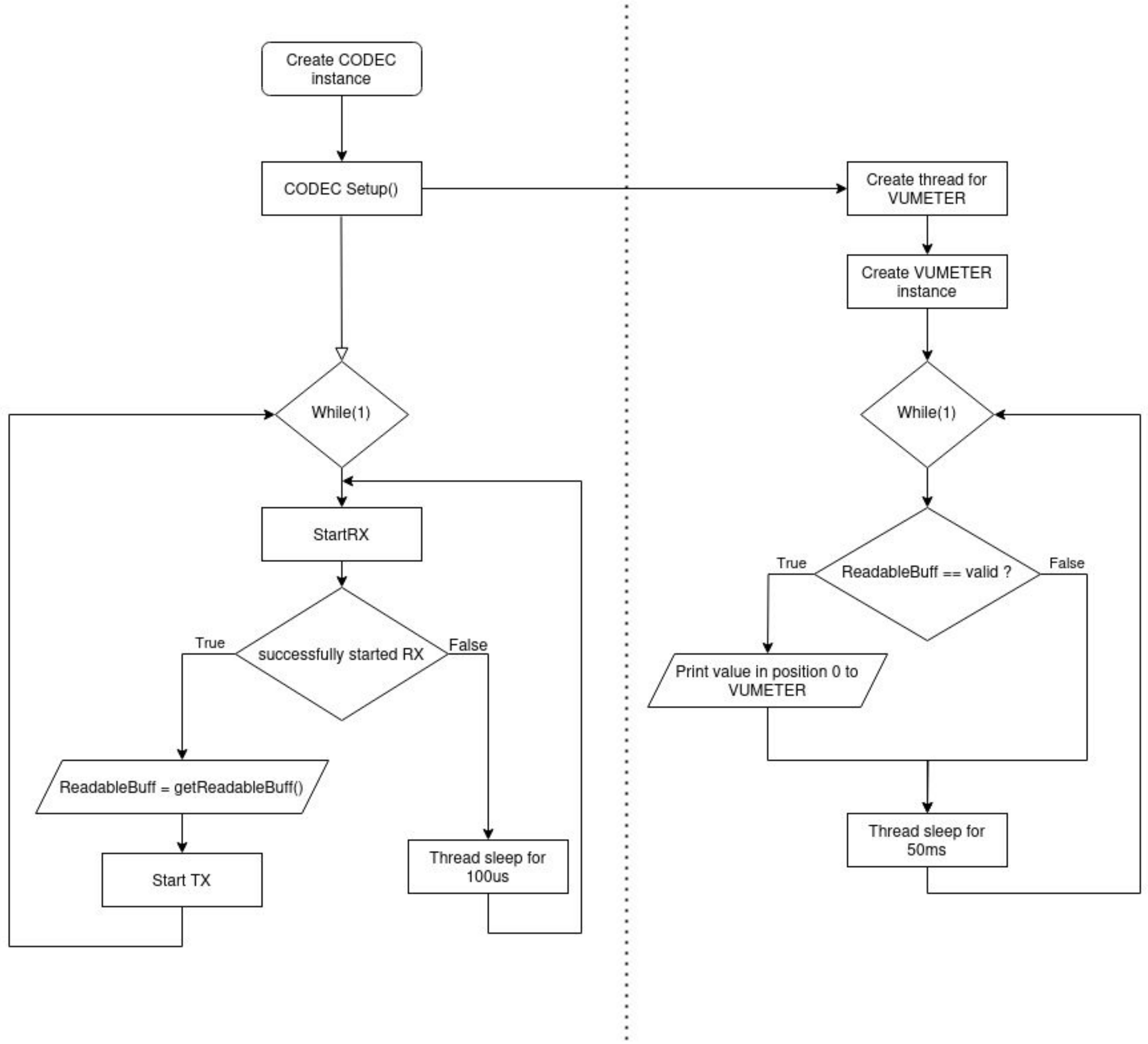
DEMO

Finally inside the “main.cpp” file we implemented a simple showcase of the project. The main idea here is:

- to create a thread that manages the VU-meter displaying the audio values when a buffer is readable
- create another thread that uses the methods described before to start I2S in RX mode and TX mode in order to implement a simple loopback - audio is immediately sent back as soon as it is received

In order to avoid that the data buffer is modified while it is being read by the VU-meter thread, a “`Lock<Mutex> lock(mutex);`” is employed

Demo flow chart


Create CODEC instance
CODEC Setup()
While(1)
StartRX
successfully started RX
True
False
ReadableBuff = getReadableBuff()
Start TX
Thread sleep for 100us
Create thread for VUMETER
Create VUMETER instance
While(1)
ReadableBuff == valid ?
True
False
Print value in position 0 to VUMETER
Thread sleep for 50ms