NC100 Model 001

AP API Specification

Amendment Record

Version	Date	Author	Reviewed	Approved	Change Description
0.9	19-4-2025	Clemens Strigl			Initial Draft
1.0	18-6-2025	Clemens Strigl			Adjusted for Client Requirements
1.1	19-6-2025	Clemens Strigl			Added Code Examples

1 Introduction

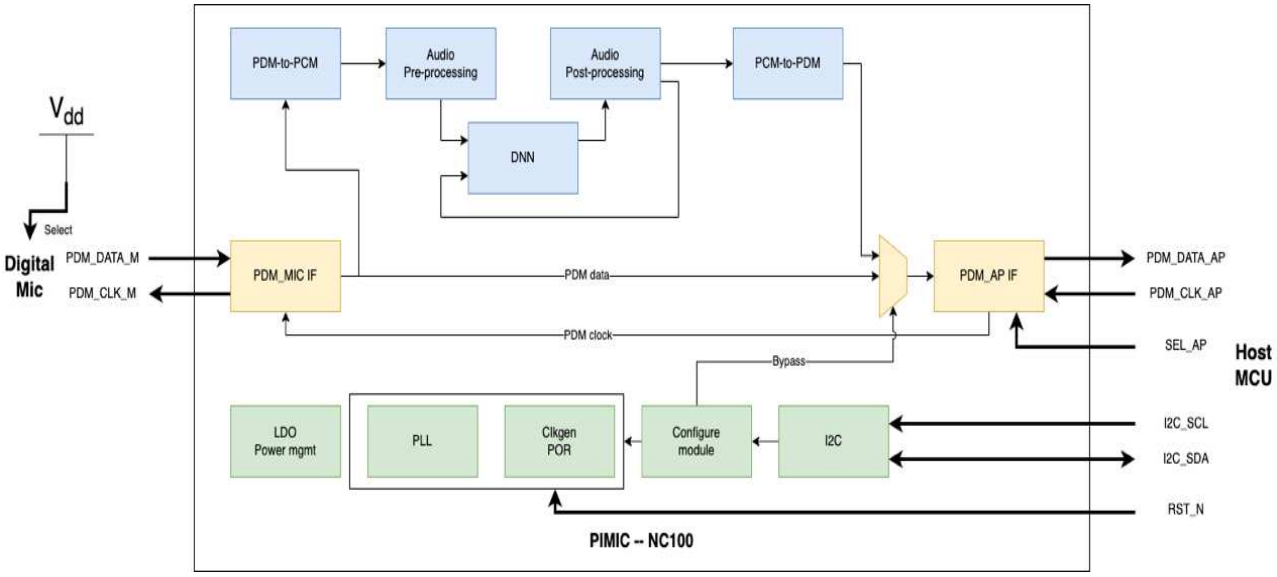
1.1 Overview

The purpose of this document is to provide the details of APIs used for communication between AP and NC100 chips. All interaction with the chips will be done and handled over the Inter-Integrated Circuit (I2C) protocol.

1.2 Abbreviation and Definitions

Abbreviations	Definitions
AP	Application Processor
I2C	Inter-Integrated Circuit
GPIO	General Purpose Input Output

2 Functional Overview



The NC100 Chip line is capable of removing noise out of an audio sample. Such noise can range from wind, keyboard noise, to airplane noise just to name a few examples. By converting PDM data to PCM thus allowing a neural network to interpreted and manipulate the input microphone data, allowing the removal of noise to be performed with a latency of around 64ms.

3 API OVERVIEW

The NC100 API library will be used to configure the NC100 Chip to specific hardware requirements that other microprocessors require when interacting with the NC100 Chip. Functionality of these setup processes are handled and communicated over the I2C protocol. All API calls are handled through the API Class object. Together with initially defining the read and write functions for this object will then assist all further functionality, thus supporting a large set of C-Based microcontrollers (such as Arduino) and allowing them to interact with the NC100 Chip.

All functions return an enumerate N100 Return Status that signals success or any issues encountered during the operations. For getter functions, all data is returned via an input pointer passed in through the function signature. Getter functions will retrieve the actual value from the chip instead of a stored value inside the class. For setter functions, all to be set values are passed in through the function signature as pass by reference. The setter functions will then automatically write to the chip register.

3.1 NC100Lib Class Initialization and Definition

```
NC100Lib (
        int resetpinNum,
        DelayMicrosecondsFunc delaymicroFunc,
        DelayMillisecondsFunc delaymillisecFunc,
        I2C_Begin begin,
        I2C_End end,
        I2C_ReadDevice readBytes,
        I2C_WriteDevice writeBytes,
        GPIOSetMode setMode,
        GPIOWrite gpiowrite,
        GPIORead gpioread
)
```

Input Function pointer described in 3.1.1-3

Input Parameter	Description
int resetpinNum	Tells NC100Lib what pin to signal on for a chip reset.
DelayMicrosecondsFunc delaymicroFunc	Defines how the microcontroller implements a delay or wait in the code. This is required to get the timing of the protocol correctly.
DelayMillisecondsFunc delaymillisecFunc	Defines how the microcontroller implements a delay or wait in the code. This is required to get the timing of the protocol correctly.
I2C_Begin begin	Begin calls the microcontroller specific I2C protocol initialization sequence, this will allow the master to talk to the slave
I2C_End end	End calls the microcontroller specific I2C protocol close or terminate command that will close all connections between the master and slave and releases all hardware allocations.
I2C_ReadDevice readBytes	ReadBytes calls the microcontroller specific I2C protocol read function. This function should define what pin the read communication is performed on. All protocol specific operations are handled by the NC100Lib, these include protocol specific timing together with pin mode switching between operations.

I2C_WriteDevice writeBytes	WriteBytes calls the microcontroller specific write function. This function should define what pin the write communication is performed on. All protocol specific operations are handled by the NC100Lib, these include protocol specific timing together with pin mode switching between operations.
GPIOSetMode setMode	SetMode sets the mode of the IO pin that is required and used for all I2C protocols. Depending on the read or write operation, this function will change the mode of the pin. These functionality is required for performing a chip reset.
GPIOWrite gpioWrite	GpioWrite calls the microcontroller specified write function. This function is not used for I2C. If communication is done directly through the GPIO instead of I2C this functions would allow such functionality. These funcitonality is required for performing a chip reset.
GPIORead gpioRead	GpioRead calls the microcontroller specified read function. This function is not used for I2C. If communcications to the chip is done over GPIO instead of I2C, this function would allow such functionality. These funcitonality is required for performing a chip reset.

3.1.1 I2C Required Function Pointers Signature

Function pointer typedefs for all I2C operations. These typedefs are used to abstract I2C functionality, allowing the implementation of these functions to vary depending on the microcontroller or specific I2C library being used. Test applications should define these functions and initialize the NC100Lib class member functions accordingly.

1. typedef en_NC100ReturnStatus (*I2C_Begin) (); Used to start or initialize the I2C communication. Returns NC100 Error Status.
2. typedef en_NC100ReturnStatus (*I2C_End) (); Used to stop or disable the I2C communication. Returns NC100 Error Status.
3. typedef en_NC100ReturnStatus (*I2C_ WriteDevice) (uint8_t address ,const uint8_t *data, size_t length); Used to write multiple bytes to the chip over the I2C protocol. This function that takes in the register address, a pointer to an array of bytes and the array length as arguments and returns NC100 Error Status.
4. typedef en_NC100ReturnStatus (*I2C_ReadDevice) (uint8_t address, uint8_t *buf, uint8_t count); Used to read data from a slave device on the I2C bus. Function that takes the address of the I2C device, the number of bytes to request, and a buffer pointer where read data is stored. It returns NC100 Error Status.

3.1.2 Timing Required Function Pointer Signature

Function pointer typedefs for all delay operations. These typedefs are used to abstract delay functionality. They are required to create protocol specific timing. Test applications should define these functions and initialize the NC100 Lib class member functions accordingly.

1. typedef void (*DelayMicrosecondsFunc) (unsigned int); Function pauses the program for a specified number of microseconds
2. typedef void (*DelayMillisecondsFunc) (unsigned int); Function pauses the program for a specified number of milliseconds

3.1.3 GPIO Required Function Pointers Signature

Function pointer typedefs for the GPIO operations. These typedefs are used to abstract GPIO functionality, allowing the implementation of these functions to vary depending on the microcontroller or specific GPIO library being used. Test applications should define these functions and initialize the NC100 Lib class member functions accordingly. These operations are used if I2C is not being used for communication between the microcontroller and the NC100 Chipset. Further, they are required for reset chip capabilities.

1. typedef en_NC100ReturnStatus (*GPIOSetMode)(uint32_t dwPin, uint32_t dwMode); Sets the mode of a specified GPIO pin. This function configures the pin to either accept input, provide output, or use an internal pull-up resistor.
2. typedef en_NC100ReturnStatus (*GPIOWrite)(uint32_t dwPin, uint32_t dwVal); Writes a digital value (HIGH or LOW) to a specified GPIO pin
3. typedef en_NC100ReturnStatus (*GPIORead)(uint32_t ulPin, int* rtn); Reads the digital value from a specified GPIO pin

3.2 Initialization

Operation Name	Parameters
NC100_chip_init	null

NC100_chip_init initializes any required initialization and is required to be called before any further getter or setter functionality of the API is called. Init initializes the I2C protocol between the AP and the Chip, thus allowing registers to be read and written to. For this function to work correctly, thought, all function pointer defined in the Class initialization have to be assigned and inputted. Without the function pointers, all further API functionality will not know how to communicate through whatever microcontroller was chosen that interacts with the NC100 Chip.

3.3 PDM Frequency

Operation Name	Default Value	Get Function	Set Function
PDM Frequency	0x03	en_NC100ReturnStatus get_PDM_freq (uint32_t* PDM_freq)	en_NC100ReturnStatus set_PDM_freq(uint32_t PDM_freq)

Getter and Setter methods for the PDM Frequency Value. This value does not correspond to the specific frequency, but an integer selection of predefined frequencies listed below:

PDM Freq Input Value	Corresponding Frequency Selected
0	768
1	1536
2	2304
3	3072

3.4 PCM Bypass Mode

Operation Name	Default Value	Get Function	Set Function
PCM Bypass	0x0	en_NC100ReturnStatus get_PCM_BYP_mode(bool* PCM_BYP_mode)	en_NC100ReturnStatus set_PCM_BYP_mode(bool PCM_BYP_mode)

Inside the chip architecture, there are multiple bypasses that are able to bypass certain feature processing steps performed on the input data. PCM Bypass enables the by passing all modules after the PCM module. This includes only the DNN filtering module also known as ENC. Setting this to true will enable the bypass.

3.5 PDM to PCM Conversion Gain Value

Operation Name	Default Value	Get Function	Set Function

PDM to PCM Gain	36	en_NC100ReturnStatus get_PDM_PCM_gain(uint32_t* gain)	en_NC100ReturnStatus set_PDM_PCM_gain(uint32_t gain)

Inside the chip architecture, PDM input data is being converted to PCM data for filtering purposes. During that conversion, extra gain can be applied programmatically. This function sets the gain value that will be applied to all input data. WARNING: audio glitches and inconsistencies may occur in the output if this value is set too high. Such glitches may occur once the value has reached over the threshold of 400.

3.6 Chip Reset

Operation Name	Input Parameters
Chip Reset	null

Chip Reset resets all register values inside the chip to their default. This includes all values capable of being modified using the NC100 API, such as the frequency and PDM2PCM gain. On the Hardware side, Low is active for the reset. No I2C programming is capable of being done if the reset pin is not hooked up and is not pulling high while not attempting to reset. Furthermore, chip init is required to be called after every reset performed to ensure that all functionality of the chip is as expected afterward.

3.7 NC100 Return Status

Error Code	Description
SUCCESS	All operations performed as expected
INVALID_PARAMETER	Input parameter does not match function signature requested data structure
I2C_INIT_ERROR	An Error occurred when attempting to initialize the I2C interface. This can either

	result from Master misconfiguration, slave misconfiguration, or an inactive slave
DATA_MISMATCH_WT_ERROR	When confirming that the data was written correctly, an error occurred because the written data and read back data does not match.
I2C_SET_LOW_FREQ_ERROR	Setting the frequency to low caused a slave error.
I2C_SET_HIGH_FREQ_ERROR	Setting the frequency to high caused a slave error.
I2C_READ_ERROR	An error occurred when attempting to read data from the slave.
I2C_WRITE_ERROR	An error occurred when attempting to write data to the slave.
GPIO_ERROR	An error occurred when attempting to communicate to the chip directly through the GPIO pins instead of I2C.

4 API Implementation Example

This C based library is very easily implement by providing the necessary function pointers for functions that act as wrappers from the I2C platform specific interface. In the following example, the platform chosen is a C based microcontroller, the Arduino Due.

4.1 Class Initialization

Quite a few functions pointers are required for the proper initialization of the NC100 Lib class. Starting off is the GPIO pin identifier which is hooked up to the NC100 reset pin is also required to be provided to the class upon initialization in the form of an integer. All other inputs for the library are platform specific function pointers.

4.1.1 Delay function pointers

The Milli- and Microsecond delay function pointers are a crucial part of the initialization. They allow the library to send out time specific bits over the GPIO pins. These are simple wrappers for the platform specific delay function, such as the ones used on the Arduino:

```
// Wrapper function for Arduino's delayMicroseconds void
arduino_delayMicroseconds(unsigned int us) {
delayMicroseconds(us);
}
// Wrapper function for Arduino's delayMilliseconds
void arduino_delaymilliSeconds(unsigned int s) {
delay(s);
}
```

4.1.2 GPIO Function Pointers

These set of function pointer dictate the defining of the pin functionality, such as input or output, together with the read and write operations of any provided pins, in the case of this API being the reset pin. Each microcontroller will have its own defines that might require a further, higher-level wrapper to fully interface with the NC100 library:

```
// Wrapper function for Arduino's GPIO functions en_NC100ReturnStatus
gpio_setMode(uint32_t dwPin, uint32_t dwMode ){
pinMode(dwPin,dwMode);
return en_NC100ReturnStatus::SUCCESS;
 }

en_NC100ReturnStatus gpio_write(uint32_t dwPin, uint32_t dwVal ){
digitalWrite(dwPin,dwVal);
return en_NC100ReturnStatus::SUCCESS;
}

int gpio_read(uint32_t dwPin ){
return digitalRead(dwPin);
}
```

4.1.3 I2C Function Pointers

These function Pointers interact with the device specific I2C library that allows the microcontroller to communicate with any I2C supportive peripheral, in this case the NC100 chip. These function include the start and end functions designed to initialize all microcontroller hardware requirements for the I2C protocol to work properly. The final two function pointers that will also be required for the NC100 Lib initialization are the read and write functions for the I2C protocol. These simply require the input of the device slave id, the buffer that will be written or read from that specific slave, and the length of the buffer.

```
// Wrapper functions for I2C operations using Wire or I2C_PORT

en_NC100ReturnStatus i2c_begin() {

I2C_PORT.begin();

return en_NC100ReturnStatus::SUCCESS;

}


en_NC100ReturnStatus i2c_end() {

I2C_PORT.end();

return en_NC100ReturnStatus::SUCCESS;

}


en_NC100ReturnStatus i2c_writeDevice(uint8_t address, const uint8_t *buf, size_t count){

I2C_PORT.beginTransmission(address);
ret = I2C_PORT.write(buf,count);
if(ret != count){
  I2C_PORT.endTransmission(true);
  return en_NC100ReturnStatus::I2C_WRITE_ERROR;
}

int stopRet = I2C_PORT.endTransmission(true);
if(stopRet != 0){
  return en_NC100ReturnStatus::I2C_WRITE_ERROR; // Or a more specific error
```

```
}
return en_NC100ReturnStatus::SUCCESS;
}


en_NC100ReturnStatus i2c_readDevice(uint8_t address, uint8_t *buf, uint8_t count) {

I2C_PORT.requestFrom(address, count, (uint8_t)true); // Request bytes

int index = 0;
while (I2C_PORT.available() && index < count) {
   uint8_t rtn = I2C_PORT.read();
   buf[index] = rtn;
   index++;
}

if (index < count) {
   return en_NC100ReturnStatus::I2C_READ_ERROR;
}
return en_NC100ReturnStatus::SUCCESS;
 }
```

4.1.4 Final Class Object Declaration

Once all platform specific function wrappers have been defined, the NC100Lib is finally able to be instantiated using the conventional C++ class constructor call:

```
//Class object creation
NC100Lib pimicLib(
  RESET_PIN,
  arduino_delayMicroseconds,
  arduino_delaymilliSeconds,
  i2c_begin,
  i2c_end,
  i2c_readDevice,
  i2c_writeDevice,
  gpio_setMode,
  gpio_write,
  gpio_read
  );
```

4.2 Getter Setter Examples

There are a few getter and setter functions exposed by the API. These include the manipulation of the PDM Frequency, ENC or Bypass Mode, and the PDM2PCM Gain register values. After class instantiation, these are able to simply called using the conventional class inherited function calling:

```
//Set PDM Frequency to be default of 3.072MHz

uint32_t new_PDM_Freq = 0x3;

en_NC100ReturnStatus  i2cRet = pimicLib .set_PDM_freq( new_PDM_Freq );

//Get PDM Frequency

uint32_t PDM_Freq;

en_NC100ReturnStatus  i2cRet = pimicLib .get_PDM_freq( &PDM_Freq );
```