**1. Voice data structures**

// Voice encode mode

// Due to a hardware bug, the actual configuration is the following

// VOICE\_ENCODE\_PCMA = 0: invalid

// VOICE\_ENCODE\_PCMU = 1: CVSD

// VOICE\_ENCODE\_CVSD = 2: PCM-u

// VOICE\_ENCODE\_BYP = 3: not compressed raw data

// It will be fixed in A2 revision back to the its originally intended configuration

typedef enum {

VOICE\_ENCODE\_PCMA = 0,

VOICE\_ENCODE\_PCMU = 1,

VOICE\_ENCODE\_CVSD = 2,

VOICE\_ENCODE\_BYP = 3

}VOICE\_ENCODE\_t;

// Voice sample rate

typedef enum {

VOICE\_RATE\_64K = 0,

VOICE\_RATE\_32K = 1,

VOICE\_RATE\_16K = 2,

VOICE\_RATE\_8K = 3

}VOICE\_RATE\_t;

// Voice notch filter configuration

typedef enum {

VOICE\_NOTCH\_BYP = 0,

VOICE\_NOTCH\_1 = 1,

VOICE\_NOTCH\_2 = 2,

VOICE\_NOTCH\_3 = 3

}VOICE\_NOTCH\_t;

// Voice polarity selection

typedef enum {

VOICE\_POLARITY\_POS = 0,

VOICE\_POLARITY\_NEG = 1

}VOICE\_POLARITY\_t;

enum{

HAL\_VOICE\_EVT\_DATA = 1,

HAL\_VOICE\_EVT\_FAIL = 0xff

};

// Voice configuration structure

typedef struct \_voice\_Cfg\_t{

bool voiceSelAmicDmic;

GPIO\_Pin\_e dmicDataPin;

GPIO\_Pin\_e dmicClkPin;

uint8\_t amicGain;

uint8\_t voiceGain;

VOICE\_ENCODE\_t voiceEncodeMode;

VOICE\_RATE\_t voiceRate;

bool voiceAutoMuteOnOff;

}voice\_Cfg\_t;

// Voice event structure

typedef struct \_voice\_Evt\_t{

int type;

uint32\_t\* data;

uint32\_t size;

}voice\_Evt\_t;

typedef void (\*voice\_Hdl\_t)(voice\_Evt\_t\* pev);

// Voice context structure

typedef struct \_voice\_Contex\_t{

bool enable;

voice\_Cfg\_t cfg;

voice\_Hdl\_t evt\_handler;

}voice\_Ctx\_t;

**2. Voice API functions**

// Enable voice core

void hal\_voice\_enable(void);

// Disable voice core

void hal\_voice\_disable(void);

// Select DMIC

void hal\_voice\_dmic\_mode(void);

// Select AMIC

void hal\_voice\_amic\_mode(void);

// Open a GPIO pin for DMIC

void hal\_voice\_dmic\_open(GPIO\_Pin\_e dmicDataPin, GPIO\_Pin\_e dmicClkPin);

// Set PGA gain for AMIC

void hal\_voice\_amic\_gain(uint8\_t amicGain);

// Set voice process gain

void hal\_voice\_gain(uint8\_t voiceGain);

// Set voice encoding mode

void hal\_voice\_encode(VOICE\_ENCODE\_t voiceEncodeMode);

// Set voice data rate

void hal\_voice\_rate(VOICE\_RATE\_t voiceRate);

// Enable voice auto-mute

void hal\_voice\_amute\_on(void);

// Disable voice auto-mute

void hal\_voice\_amute\_off(void);

// Voice interrupt handler

void \_\_attribute\_\_((weak)) hal\_ADC\_IRQHandler(void);

// Allocate memory and power manager for voice

void hal\_voice\_init(void);

// Configure voice capture

int hal\_voice\_config(voice\_Cfg\_t cfg, voice\_Hdl\_t evt\_handler);

// Start voice capture

int hal\_voice\_start(void);

// Stop voice capture

int hal\_voice\_stop(void);

// Clear memory and power manager for voice

int hal\_voice\_clear(void);

**3. Voice capture application examples**

**3.1. Variables for voice capture with voice buffer**

#define MAX\_VOICE\_BUF\_SIZE 24576

static uint8 voiceDemo\_TaskID; // Task ID for internal task/event processing

static int voiceWriteID;

static uint32\_t voiceBuf[MAX\_VOICE\_BUF\_SIZE];

static int voiceBufIndex;

**3.2. Voice event handler with voice buffer, read voice data and write to voice buffer**

static void voice\_evt\_handler\_buf(voice\_Evt\_t \*pev)

{

if(pev->type == HAL\_VOICE\_EVT\_DATA){

// LOG("Type matched\n");

int n;

for (n = 0; n < pev->size; n++) {

voiceBuf[voiceBufIndex] = pev->data[n];

voiceBufIndex++;

}

voiceWriteID++;

LOG("Voice data saved successfully:");

LOG("%d %d\n", voiceWriteID, pev->data[pev->size-1]);

}

}

**3.3. Configure and start voice capture task**

static void voiceCaptureTask( void )

{

voice\_Cfg\_t cfg;

cfg.voiceSelAmicDmic = 1;

cfg.dmicDataPin = P4;

cfg.dmicClkPin = P5;

cfg.amicGain = 0;

cfg.voiceGain = 40;

cfg.voiceEncodeMode = VOICE\_ENCODE\_CVSD;

cfg.voiceRate = VOICE\_RATE\_8K;

cfg.voiceAutoMuteOnOff = 1;

volatile int voiceConfigStatus = hal\_voice\_config(cfg, voice\_evt\_handler\_buf);

if(voiceConfigStatus) {

LOG("Voice configuration failed\n");

return;

}

else {

LOG("Voice configuration succeeded\n");

}

LOG("Voice capture started\n");

// Restart timer

osal\_start\_timerEx( voiceDemo\_TaskID, voiceStopTask\_EVT, 5000 );

hal\_voice\_start();

}

**3.4. Stop voice capture task**

// This function will print the voice buffer data via UART

static void voiceStopTask( void )

{

hal\_voice\_stop();

LOG("Voice capture stopped\n");

LOG("Voice data word saved:");

LOG("%d\n", voiceBufIndex);

if (hal\_uart\_get\_tx\_ready() == PPlus\_SUCCESS) {

int voiceTxmitIndex;

voiceTxmitIndex = 0;

while (voiceTxmitIndex < voiceBufIndex) {

LOG("%x\n",voiceBuf[voiceTxmitIndex]);

voiceTxmitIndex++;

}

LOG("Voice data byte transmitted:");

LOG("%d\n", voiceTxmitIndex);

LOG("Voice data buffer transmitted successfully");

}

}

**3.5. Initialize voice buffer pointers and start voice capture**

void voice\_Init( uint8 task\_id )

{

voiceDemo\_TaskID = task\_id;

voiceWriteID = 0;

voiceBufIndex = 0;

voiceCaptureTask();

}

**3.6. Voice event process**

uint16 voice\_ProcessEvent( uint8 task\_id, uint16 events )

{

VOID task\_id; // OSAL required parameter that isn't used in this function

LOG("voice\_ProcessEvent: 0x%x\n",events);

if ( events & SYS\_EVENT\_MSG )

{

uint8 \*pMsg;

if ( (pMsg = osal\_msg\_receive( voiceDemo\_TaskID )) != NULL )

{

voice\_ProcessOSALMsg( (osal\_event\_hdr\_t \*)pMsg );

// Release the OSAL message

VOID osal\_msg\_deallocate( pMsg );

}

// return unprocessed events

return (events ^ SYS\_EVENT\_MSG);

}

if ( events == voiceCaptureTask\_EVT )

{

LOG("voiceCaptureTask\_EVT\n");

voiceCaptureTask();

return (events ^ voiceCaptureTask\_EVT);

}

if ( events == voiceStopTask\_EVT )

{

// Perform periodic heart rate task

LOG("voiceStopTask\_EVT\n");

voiceStopTask();

return (events ^ voiceStopTask\_EVT);

}

// Discard unknown events

return 0;

}

**3.7. OSAL initialization for voice applications**

const pTaskEventHandlerFn tasksArr[] =

{

LL\_ProcessEvent,

voice\_ProcessEvent

};

void osalInitTasks( void )

{

uint8 taskID = 0;

tasksEvents = (uint16 \*)osal\_mem\_alloc( sizeof( uint16 ) \* tasksCnt);

osal\_memset( tasksEvents, 0, (sizeof( uint16 ) \* tasksCnt));

LL\_Init( taskID++ );

/\* Application \*/

voice\_Init( taskID );

}

**4. Matlab scripts for captured voice data processing and playback**

Process and playback raw voice data: pcmraw.m

Process and playback PCM-u coded voice data: pcmu.m

Process and playback PCM-A coded voice data: pcma.m

Process and playback CVSD coded voice data: cvsd.m