OCT6100 API

(Application Programming Interface) Specification

Revision 4.0

The OCT6100 API provides a layer of software for integrating the OCT6100 Echo Cancellation device into customer designs. It provides a flexible architecture that will integrate easily into several different target OS models. The API allows the user to access all the flexibility and features of the OCT6100 device from device initialization to configuring and controlling voice streams.

Octasic Inc.
4101 Molson St., Suite 300
Montreal, QC H1Y 3L1
Canada

www.octasic.com

1 Introduction

This document defines the C-language application programming interface (API) designed to control the OCT6100 Echo Cancellation device. The API, provided by Octasic, enables the user to quickly exploit the features offered by the OCT6100.

1.1 OCT6100 Product family

The OCT6100 represents a family of full-featured 128 ms tail G.168 (2004) compliant voice processing devices with varying features and capacities. For an overview of all features, and for a listing of specific devices and included features see appendix A of the HW specification.

Some of the available telephony functions performed by OCT6100 devices include: ADPCM compression/decompression, Automatic Level Control (ALC), Adaptive Noise Reduction (ANR), buffer play-out, tone detection, conferencing and echo cancellation on up to 672 channels. All channels can be individually configured in either electric echo cancellation or acoustic echo suppression modes.

This document describes the software API that allows integration of the OCT612x family of devices. The OCT612x is the most complete line of voice processing device and provides echo cancellation (Acoustic or Line), Automatic Level Control and Noise reduction. Furthermore, it provides advanced voice processing gateway features such as buffer play-out for announcements and tones, extensive signaling tone detection, conferencing and ADPCM compression/decompression.

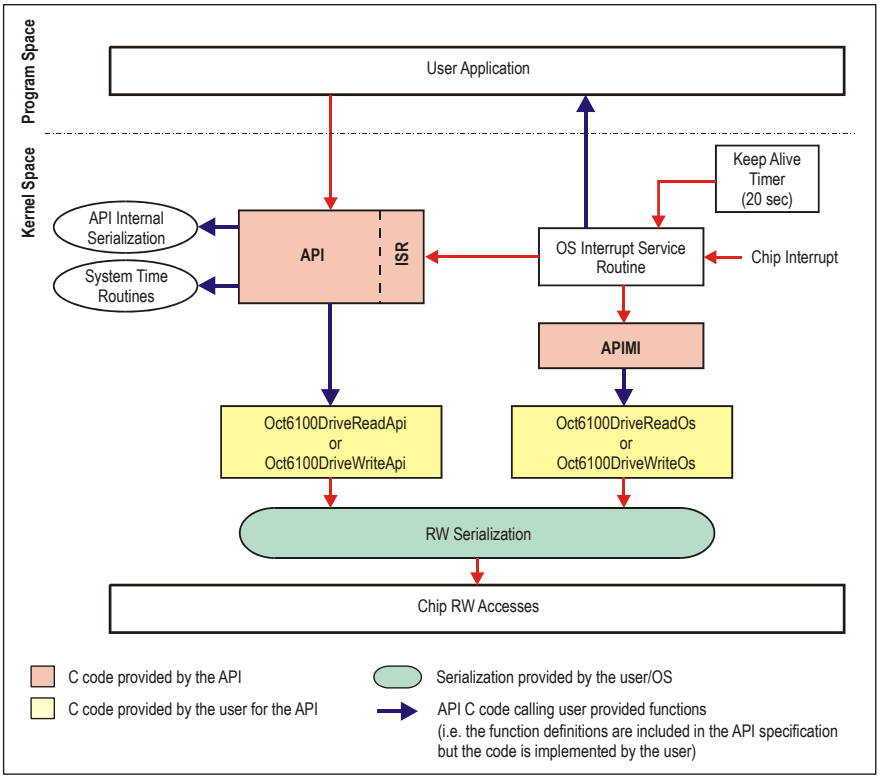
2 System Architecture

The API code is designed stateless to allow one set of code to manage multiple devices. All information regarding a chip serviced by the API is kept in a contiguous block of shared memory allocated by the user. This shared memory is referred to as the chip's instance structure. This permits the API code to service multiple chips within the host system. The instance structure has a one-to-one correspondence with a chip, and thus a pointer to it is the first parameter for all function calls provided by the API.

To allow the API to identify a chip to the user software, the user provides a unique value to the **Oct6100ChipOpen** function (parameter **ulUserChipId**). This value is the identifier of the chip within the user's system. The value is not directly used or interpreted by the API: it is stored within the chip's instance structure. When an I/O access must be performed, the API provides the chip ID as a parameter to the I/O routine. This allows the user's I/O driver to access the proper device.

The figure below depicts an example of an interrupt driven system architecture servicing multiple chips. The interrupts do not use deferred procedure calls. All API functions are contained in the API block, with the exception of one function (**Oct6100InterruptMask**).

Because the masking of the chip's interrupt pin is typically carried out at a priority level higher than the call of the API's ISR, the **Oct6100InterruptMask** function is contained in an independent code block labeled APIMI. The user must write separate R/W user functions for APIMI block.



System Architecture without Deferred Interrupt Procedure Call

In a Deferred Interrupt implementation with multiple OCT6100 devices, the user may choose to use the chip ID as an index. The pointer to the instance structure of each chip can be kept in a global array indexed by chip ID. The host application can thus refer to a chip as an index, and use that index to obtain the instance structure when an API function must be called.

3 Software Architecture

This section presents the API software architecture as well as how it interacts with the host system, and in particular, how interrupts are handled by the API.

3.1 API Architecture

The API architecture section is broken down into five parts:

- Instance Structure
- Default Functions and Function Structures
- Serialization
- Multi-Process System
- Example code

Each part is described in the sections below.

3.1.1 Instance Structure

The API code is stateless. This allows the same code to service multiple chips in the host system. All information regarding the state of a chip serviced by the API is kept in an instance structure.

The instance structure is a block of contiguous memory allocated by the user. When the chip is configured via the **OCT6100ChipOpen** API function, the instance structure is initialized.

A pointer to the instance structure's memory is passed to each API function as a parameter. As API functions are called to open and close voice streams, the instance structure is updated to reflect the current state of the chip.

This structure is managed entirely by the API. It is not intended to be interpreted by the user.

3.1.2 Function Structures and Default Functions

Each API function is associated with a parameter structure and default function. The parameter structure contains all user parameters needed by an API function to perform the required task. Both the function and the default function have a pointer to this structure as an argument.

These parameter structures are a vehicle used to pass parameters to the API. These structures can be created locally and then destroyed after an API call completes.

The default functions allow for forward-compatibility of enhanced functionality, and offers default settings for most parameters.

The names of the function, default function and structure respect a standard convention within the API. For a function called "Example" the identifiers would be:

- API function: *Oct6100ExampleFunction*
- Associated default function *Oct6100ExampleFunctionDef*
- Associated function structure *tOCT6100_EXAMPLE_FUNCTION*

To demonstrate the use of the default functions, refer to the example provided in the **Sample Code** section.

NOTE: The use of the default functions is not necessary but is strongly recommended to ensure backward compatibility of user code with future releases of an API.

3.1.3 Serialization

Serialization is required for I/O accesses to the chip, and must be provided by the user's I/O driver. Serialization is needed because the chip's internal registers and memories are accessed via indirection registers. All I/O routines provided by the user must use a common serialization object to ensure that API and APIMI R/W operations to the chip are atomic.

3.1.4 Multi-Process System

The user must indicate to the API if the host system is a multi-process one or not, via the **fMultiProcessSystem** flag. For a single-process system all serialization object handles are created by the API **Oct6100OpenChip** function and kept internal to the API instance. There is no need for the user to create a local instance structure.

For a multi-process system the main process must perform the following steps:

- The **Oct6100ChipOpenDef** function is called and the structure is modified to obtain the desired chip configuration. At this point, the **fMultiProcessSystem** flag should be set to TRUE.
- The size needed for the API instance is obtained via the **Oct6100GetInstanceSize** function.
- A shared memory block (pApiInstShared) of the returned size is allocated using the tPOCT6100_INSTANCE_API type and stored in an area such that all processes can share and access it.
- A local tOCT6100_INSTANCE_API structure (ApiInstLocal) is created and kept in the user's process space. This structure is of the same type as the shared instance structure memory just allocated, but not the same size. The structure will contain the portion of the API that is process specific (such as serialization object handles). This structure is small enough to be kept on the stack. It does not have to be allocated dynamically.
- The API function **Oct6100CreateLocalInstance** is called. This will initialize all process specific portions of the local API instance structure (ApiInstLocal). Also, a pointer to the shared portion of the API instance (pApiInstShared) will be stored in the local structure.

- The function **Oct6100ChipOpen** is called with the local API instance structure (ApiInstLocal) as its first argument.
- All other API function calls will have the local instance structure as the first parameter.

For every subsequent process which will access the chip:

- A local tOCT6100_INSTANCE_API structure (ApiInstLocal) is created and kept in the user's process space. The user's process is responsible for connecting to the shared memory allocated by the main process (pApiInstShared).
- The API function **Oct6100CreateLocalInstance** is called. This will initialize all process specific portions of the local API instance structure (ApiInstLocal). Also, a pointer to the shared portion of the API instance (pApiInstShared) will be stored in the local structure.
- All API function calls are performed with the local instance structure (ApiInstLocal) as the first argument.

Example code demonstrating these steps is contained in the next section.

3.1.5 Sample Code

The typical usage of functions respects the following sequence:

- A parameter structure is allocated.
- The Default function is called. Default configuration functions are identified by the "Def" suffix at the end of the function name.
- The user changes the appropriate parameters in the structure as required.
- The actual function is called.

The example below illustrates the chip initialization sequence in a single-process system:

```
#include "oct6100_api.h"
void main( )
{     tPOCT6100_INSTANCE_API          pApiInstance;
      tOCT6100_CHIP_OPEN              ChipOpen;
      tOCT6100_GET_INSTANCE_SIZE      InstanceSize;
      tOCT6100_CHIP_STATS             ChipStats;
      tOCT6100_CHIP_CLOSE             CloseChip;
      UINT32                          ulResult;
      unsigned char * pbyFileData = NULL;
      FILE* pFile;
      int iLen;

      /* Inserting default values into structure configuration parameters. */
      Oct6100ChipOpenDef ( &ChipOpen );

      /* Change default parameters as needed (e.g. changing the clock frequency). */
      ChipOpen.ulUpclkFreq = cOCT6100_UPCLK_FREQ_33_33_MHZ;
      OpenChip.fEnableMemClkOut = TRUE;
      OpenChip.ulMemClkFreq = cOCT6100_MCLK_FREQ_133_MHZ;
      OpenChip.ulUserChipId = 1;
      OpenChip.ulMemoryType = cOCT6100_MEM_TYPE_DDR;
      OpenChip.ulNumMemoryChips = 2;
      OpenChip.ulMemoryChipSize = cOCT6100_MEMORY_CHIP_SIZE_32MB;

      /* Load the image file */
      /* Open the file.*/
      pFile = fopen( "oct6100.ima", "rb" );
      if ( pFile == NULL )
      {
            printf( "fopen\n" );
            exit( EXIT_FAILURE );
      }

      /* Get the file length.*/
      fseek( pFile, 0L, SEEK_END );
      iLen = ftell( pFile );
      fseek( pFile, 0L, SEEK_SET );

      /* Allocate enough memory to store the file content.*/
      pbyFileData = (unsigned char *)malloc( iLen );
      if ( pbyFileData == NULL )
      {
            fclose( pFile );
            printf( "malloc\n" );
            exit( EXIT_FAILURE );
      }

      /* Read the content of the file.*/
      fread( pbyFileData, 1, iLen, pFile );

      /* The content of the file should now be in the pbyFileData memory, we can */
      /* close the file and return to the calling function.*/
      fclose( pFile );

      /* Assign the image file.*/
      OpenChip.pbyImageFile = pbyFileData;
      OpenChip.ulImageSize = iLen;

      /* Inserting default values into tOCT6100_GET_INSTANCE_SIZE structure parameters. */
      Oct6100GetInstanceSizeDef ( &InstanceSize );

      /* Get the size of the OCT6100 instance structure. */
      ulResult = Oct6100GetInstanceSize ( &ChipOpen, &InstanceSize );
      if (ulResult != cOCT6100_ERR_OK)
      {
            /* Error handling. */
```

```
    }

    /* Allocate memory for the API Instance structure */
    pApiInstance = (tPOCT6100_INSTANCE_API) malloc(InstanceSize.ulApiInstanceSize);
    if ( pApiInstance == NULL )
    {
        /* Error handling. */
    }

    /* Perform the actual configuration of the chip. */
    ulResult = Oct6100ChipOpen ( pApiInstance, &ChipOpen );
    if (ulResult != cOCT6100_ERR_OK)
    {
        /* Error handling. */
    }

    /* Done with the firmware memory. */
    free( pbyFileData );
    pbyFileData = NULL;

    /* The chip is running..  Enter the main loop (where channels are opened and voice is processed).. */
    /* For this example, nothing much is done except getting chip stats */

    printf( "Chip is running – Getting chip stats... \n" );

    Oct6100ChipGetStatsDef ( &ChipStats );
    ChipStats.fResetChipStats = FALSE;

    while ( fWhileCondition == TRUE )
    {
        ulResult = Oct6100ChipGetStats(pApiInstance, &ChipStats );
        if ( cOCT6100_ERR_OK != ulResult )
        {
            printf( "Oct6100ChipGetStats\n" );
            exit( EXIT_FAILURE );
        }


        printf( "Current number of opened channels is %d\r",
                                                        ChipStats.ulNumberChannels );

        /* For demonstration purposes, this loop is only executed once. */
        printf( "\nDone!" );
        break;
    }

    /* Close access to the chip, we are done with it. */
    Oct6100ChipCloseDef( &CloseChip );

    ulResult = Oct6100ChipClose(pApiInstance, &CloseChip );
    if ( cOCT6100_ERR_OK != ulResult )
    {
        printf( "Oct6100ChipClose\n" );
        exit( EXIT_FAILURE );
    }


}
```

The next example code demonstrates the chip initialization sequence in a multi-process system. Here is the code that would initialize the primary/master process:

```
#include "oct6100_api.h"

int main( int argc, char *argv[ ] )
{
	unsigned char * pbyFileData = NULL;
	FILE* pFile;
	int iRes;
	int iLen;
	int fWhileCondition = TRUE;

	UINT32 ulResult;

	tOCT6100_CHIP_OPEN                  OpenChip;
	tOCT6100_GET_INSTANCE_SIZE          GetInstanceSizeParms;
	tPOCT6100_INSTANCE_API              pApiInstanceShared;
	tOCT6100_CHIP_STATS                 ChipStats;
	tOCT6100_CREATE_LOCAL_INSTANCE      CreateLocal;
	tOCT6100_DESTROY_LOCAL_INSTANCE     DestroyLocal;
	tOCT6100_CHIP_CLOSE                 CloseChip;
	tOCT6100_INSTANCE_API               ApiInstLocal;

	/* First set default values of OpenChip. */
	Oct6100ChipOpenDef( &OpenChip );

	/* Set the general parameters. */
	OpenChip.ulUpclkFreq = cOCT6100_UPCLK_FREQ_33_33_MHZ;
	OpenChip.fEnableMemClkOut = TRUE;
	OpenChip.ulMemClkFreq = cOCT6100_MCLK_FREQ_133_MHZ;
	OpenChip.ulUserChipId = 1;
	OpenChip.ulMemoryType = cOCT6100_MEM_TYPE_DDR;
	OpenChip.ulNumMemoryChips = 2;
	OpenChip.ulMemoryChipSize = cOCT6100_MEMORY_CHIP_SIZE_32MB;

	/* Set the MULTI-PROCESS flag. */
	OpenChip.fMultiProcessSystem = TRUE;

	/* Load the image file */
	/* Open the file.*/
	pFile = fopen( "oct6100.ima", "rb" );
	if ( pFile == NULL )
	{
		printf( "fopen\n" );
		exit( EXIT_FAILURE );
	}

	/* Get the file length.*/
	fseek( pFile, 0L, SEEK_END );
	iLen = ftell( pFile );
	fseek( pFile, 0L, SEEK_SET );

	/* Allocate enough memory to store the file content.*/
	pbyFileData = (unsigned char *)malloc( iLen );
	if ( pbyFileData == NULL )
	{
		fclose( pFile );
		printf( "malloc\n" );
		exit( EXIT_FAILURE );
	}

	/* Read the content of the file.*/
	fread( pbyFileData, 1, iLen, pFile );

	/* The content of the file should now be in the pbyFileData memory, we can */
	/* close the file and return to the calling function.*/
	fclose( pFile );

	/* Assign the image file.*/
```

```
OpenChip.pbyImageFile = pbyFileData;
OpenChip.ulImageSize = iLen;

/* Get the instance size. */
Oct6100GetInstanceSizeDef( &GetInstanceSizeParms );

ulResult = Oct6100GetInstanceSize( &OpenChip, &GetInstanceSizeParms );

if ( cOCT6100_ERR_OK != ulResult )
{
      printf( "Oct6100GetInstanceSize\n" );
      exit( EXIT_FAILURE );
}

/* Allocate instance memory. */
/* The main process is responsible for allocating the shared memory. */
/* Replace the following function call with your OS specific shared memory */
/* "allocation" function. */

pApiInstanceShared = (tPOCT6100_INSTANCE_API)OsSpecificSharedMemoryAlloc(
                                                      "SHARED_MEM_OCT6100",
                                                      GetInstanceSizeParms.ulApiInstanceSize );
if( !pApiInstanceShared )
{
      printf( "OsSpecificSharedMemoryAlloc\n" );
      exit( EXIT_FAILURE );
}

/* Create the local instance portion. */
Oct6100CreateLocalInstanceDef( &CreateLocal );

CreateLocal.pApiInstShared = pApiInstanceShared;
CreateLocal.pApiInstLocal = &ApiInstLocal;
CreateLocal.pProcessContext = USER_DEFINED_PRIMARY_IDENTIFIER;
CreateLocal.ulUserChipId = OpenChip.ulUserChipId;

ulResult = Oct6100CreateLocalInstance( &CreateLocal );
if ( cOCT6100_ERR_OK != ulResult )
{
      printf( "Oct6100CreateLocalInstance\n" );
      exit( EXIT_FAILURE );
}

/* Perform actual open of chip */
ulResult = Oct6100ChipOpen( &ApiInstLocal, &OpenChip );
if ( cOCT6100_ERR_OK != ulResult )
{
      printf( "Oct6100ChipOpen\n" );
      exit( EXIT_FAILURE );
}

/* Done with the firmware memory. */
free( pbyFileData );
pbyFileData = NULL;

/* The chip is running..  Enter the main loop.. */

/* For this example, nothing much is done except getting chip stats while */
/* waiting for the child process to connect. */

printf( "Chip is running -- Waiting for child process to connect... \n" );

Oct6100ChipGetStatsDef ( &ChipStats );
ChipStats.fResetChipStats = FALSE;

while ( fWhileCondition == TRUE )
{
      ulResult = Oct6100ChipGetStats( &ApiInstLocal, &ChipStats );
      if ( cOCT6100_ERR_OK != ulResult )
      {
```

```
                printf( "Oct6100ChipGetStats\n" );
                exit( EXIT_FAILURE );
            }


            printf( "Current number of opened channels is %d\r",
                                                                ChipStats.ulNumberChannels );

            /* For demonstration purposes, this loop is only executed once. */
            printf( "\nDone!" );
            break;

            /* In a real multi-process application, we would have to wait for the child */
            /* process to land before carrying on with closing the chip. */
        }

        /* Close access to the chip, we are done with it. */
        Oct6100ChipCloseDef( &CloseChip );

        ulResult = Oct6100ChipClose( &ApiInstLocal, &CloseChip );
        if ( cOCT6100_ERR_OK != ulResult )
        {
            printf( "Oct6100ChipClose\n" );
            exit( EXIT_FAILURE );
        }

        /* Destroy the local instance after all accesses to the chips are done with. */
        Oct6100DestroyLocalInstanceDef( &DestroyLocal );

        ulResult = Oct6100DestroyLocalInstance( &ApiInstLocal, &DestroyLocal );
        if ( cOCT6100_ERR_OK != ulResult )
        {
            printf( "Oct6100DestroyLocalInstance\n" );
            exit( EXIT_FAILURE );
        }

        /* Unallocate shared memory since this is the main process. */
        /* Replace the following function call with your OS specific shared memory */
        /* "free" function. */
        iRes = OsSpecificSharedMemoryFree(
                                            pApiInstanceShared );
        if( iRes < 0 )
        {
            printf( "OsSpecificSharedMemoryFree\n" );
            exit( EXIT_FAILURE );
        }

        exit( EXIT_SUCCESS );
}
```

Code to initialize the secondary process:

```
#include "oct6100_api.h"

int main( int argc, char *argv[ ] )
{
	tPOCT6100_INSTANCE_API pApiInstanceShared;
	tOCT6100_CREATE_LOCAL_INSTANCE CreateLocal;
	tOCT6100_DESTROY_LOCAL_INSTANCE DestroyLocal;
	tOCT6100_INSTANCE_API ApiInstLocal;
	tOCT6100_INTERRUPT_FLAGS IntFlags;

	UINT32 ulResult;
	UINT32 fWhileCondition = TRUE;

	/* Connect to the shared memory allocated by the primary (master) process. */
	/* The main process is responsible for allocating the shared memory. */
	/* Replace the following function call with your OS specific shared memory */
	/* "connect" function. */
	pApiInstanceShared = (tPOCT6100_INSTANCE_API)OsSpecificSharedMemoryConnect(
                                                                    "SHARED_MEM_OCT6100" );

	/* Create the local instance portion. */
	Oct6100CreateLocalInstanceDef( &CreateLocal );

	CreateLocal.pApiInstShared = pApiInstanceShared;
	CreateLocal.pApiInstLocal = &ApiInstLocal;
	CreateLocal.pProcessContext = USER_DEFINED_SECONDARY_IDENTIFIER;
	CreateLocal.ulUserChipId = 1;

	ulResult = Oct6100CreateLocalInstance( &CreateLocal );
	if ( cOCT6100_ERR_OK != ulResult )
	{
		printf( "Oct6100CreateLocalInstance\n" );
		exit( EXIT_FAILURE );
	}

	/* We are connected to the chip..  Enter the main loop.. */

	/* One could poll for tone events, or retrieve debugging data, etc... */
	printf( "Connected to the 6100API -- Calling ISR in loop... \n" );

	while ( fWhileCondition == TRUE )
	{
		/* Call the ISR */
		Oct6100InterruptServiceRoutineDef( &IntFlags );

		ulResult = Oct6100InterruptServiceRoutine(
                                        &ApiInstLocal, &IntFlags );
		if ( cOCT6100_ERR_OK != ulResult )
		{
			printf( "Oct6100InterruptServiceRoutine\n" );
			exit( EXIT_FAILURE );
		}

		/* For demonstration purposes, this loop is only executed once. */
		printf( "\nDone!" );
		break;

		/* In a real multi-process application, the child would have to exit */
		/* somehow and let the main/master process know that it is done with */
		/* the shared memory. */
	}

	/* Destroy the local instance */
	Oct6100DestroyLocalInstanceDef( &DestroyLocal );

	ulResult = Oct6100DestroyLocalInstance( &ApiInstLocal, &DestroyLocal );
	if ( cOCT6100_ERR_OK != ulResult )
	{
```

```
            printf( "Oct6100DestroyLocalInstance\n" );
            exit( EXIT_FAILURE );
      }

      /* Disconnect from shared memory. */
      /* Replace the following function call with your OS specific shared memory */
      /* "diconnect" function. */
      OsSpecificSharedMemoryDisconnect( pApiInstanceShared );

      /* The main/master process can "free" the shared memory when required. */

      exit( EXIT_SUCCESS );
}
```

Almost all functions require a pointer to the chip's API instance structure as the first parameter. This instance structure is created by the user before the call to **Oct6100ChipOpen** and is unique to each chip being managed by the software.

The structure keeps the state of an instance of a chip and is required to perform any operations on the chip. See Section 2 - System Architecture.

3.2 Interrupts

3.2.1 Chip Configuration

The OCT6100 device features an interrupt pin that can be configured to respect a minimum time between successive interrupts.

Each individual event can be further configured to meet the host system's needs and capabilities. Each error or status report that the chip issues can be individually configured to generate an interrupt or not. The interrupts are configured at start-up time (**Oct6100ChipOpen**), and can be reconfigured during the chip operation (**Oct6100InterruptConfigure**).

Each interrupt flag can be configured in one of three ways:

- **Disable the interrupt** - The corresponding Interrupt Enable is cleared and the condition will not generate an interrupt.
- **Enable the interrupt without timeouts** - The corresponding Interrupt Enable is set. If the same interrupt occurs repeatedly, each new event occurrence will cause the interrupt pin to be asserted. This will cause the ISR to be called each time.
- **Enable the interrupt and enforce timeouts** The corresponding Interrupt Enable is set. When an interrupt is generated and the API ISR is called, the API will disable the corresponding Interrupt Enable in the device. The API will only re-enable this Interrupt Enable after the user-specified timeout period. This allows the user to mask Interrupts for long periods of time.

3.2.2 Interrupt Service Routine

The API provides an interrupt service routine (ISR), **Oct6100InterruptServiceRoutine**, to service the events reported by the chip's interrupt registers. The ISR updates chip and channel statistics maintained by the API and empties the device's tone event buffer queuing the events in API memory to be read by the user software. The functions performed by the ISR on an interrupt are determined by the current events in the interrupt vector or the elapsed time since last servicing of the resource (e.g. offloading or extending counters). The interrupt service routine can be used in either an interrupt-driven system or a polling-driven system. Note that the **Oct6100ChipOpen** function must be called successfully before the ISR can be used.

3.2.3 Interrupt-Driven System

Interrupts can be handled by the host system using one of two methods: with or without deferred procedure calls (DPC). This choice is left to the system designer.

3.2.3.1 Interrupt-Driven System without DPCs

When DPCs are not used, the API's ISR is called by the OS's ISR directly at the interrupt priority level. This means the interrupt is responded to very quickly, however, it will force the OS ISR to include the servicing of the interrupt. An example of an interrupt-driven system implemented without DPC is illustrated in the **System Architecture** section above.

In non-DPC case, calling the **Oct6100InterruptMask** function is optional, as the OCT6100 interrupt will be handled immediately.

3.2.3.2 Interrupt-Driven system with DPCs

When using DPCs, the user causes a DPC to invoke the API's ISR. In this case, the DPC causes the ISR treatment to be delayed to a later time, and typically treated at a lower priority level.

To prevent from flooding the OS with the same interrupt event, the chip's interrupt pin must be masked until the API ISR is called. The API provides the **Oct6100InterruptMask** function to mask the interrupt pin. The mask interrupt routine will mask the interrupt time for a period of 60 ms. Using this masking feature ensures convergence of the system; if the OS does not answer the signaled interrupt, the chip will reactivate its interrupt pin without software intervention. Once it is called, the API's ISR will service all flagged events and immediately re-enable the interrupt pin.

Once the interrupt is signaled to the OS, the OS queues the interrupt request to be treated at a later time. When the host software decides to service the deferred interrupt, the API's **Oct6100InterruptServiceRoutine** function determines which error(s) or event(s) generated the interrupt, and the API then performs the necessary actions to fix the error(s) or service the event(s). Before the API's ISR function exits, the chip's interrupt pin is re-enabled.

3.2.4 Interrupt Polling-Driven System

In the case of an interrupt polling-driven system, the chip's interrupt pin is not used: The API's ISR must be called periodically by the host application. Also, because the API's ISR is responsible for updating the software extension of certain chip statistics, the API's ISR must be called a minimum of every 20 seconds to ensure that the integrity of these statistics is maintained. Failure to do so will not impair the operational call integrity but may cause some statistics to be invalid.

4 Using the OCT6100 API

4.1 Definitions

Channel	A channel is composed of 2 TDM voice streams.
TSST	A TDM time-slot stream. A specific timeslot on a specific stream of the TDM bus.
Conference Bridge	Resource used to mix multiple voice streams.
Tone Detector	Chip resource used to detect signaling tones on a voice stream.
SIN Port	Hybrid side Send Input connection interface (near-end input).
SOUT Port	Network side Send Output connection interface (far-end output)
RIN Port	Network side Receive Input connection interface (far-end input).
ROUT Port	Hybrid side Receive Output connection interface (near-end output).

4.2 Documentation and Coding Conventions

In this document:

- All addresses are byte addresses.
- Numbers are decimal unless otherwise specified.
- A word is 16 bits, and a byte is 8 bits.
- All memory locations are laid-out in the big-endian format.
- When a parameter value is greater than 32 bits, it is stored in an array where the lowest indexed element contains the LSB.

All function parameters are passed in C structures to allow for compatibility of code upgrades. Each parameter is documented here with 3 fields:

Direction – Indicates if the parameter is an input (IN), output (OUT), or input and output (IO) of the function. When a parameter is a pointer, the direction is indicated as direction/direction, where the first direction refers to the pointer itself (typically IN) and the second direction (after the slash) refers to the memory pointed to by the pointer. Thus, an IN/OUT pointer direction indicates that the pointer is an input to the function (i.e. the value of the pointer will not be modified), and the memory pointed to by the pointer is used for output.

Type – Indicates the C type of the parameter. A UINT32 is an unsigned 32-bit value. Parameters may also be declared as arrays and are documented here as UINT32[*x*] where *x* indicates the number of elements. Also used in the API are unsigned characters (8-bit values) indicated as BYTEs. As with ULONGs, parameters may also be declared as arrays and are documented here as UINT8[*x*] where *x* indicates the number of elements.

Default – Indicates the default value to which the parameter is set by an associated function for initializing the structure. All values of OCT6100_INVALID*Xxx* means that the Def function will initialize the parameter to a value which indicates invalid for that parameter. The API will return an error if the parameter remains invalid when the structure is passed to a function that uses the parameter as an input.

4.2.1 Return Values

All API functions return the OCT6100_RC_OK value when they complete successfully. Generally, functions return non-successful indications when there is an improper usage of the API or device. For example, conflicting parameters or exceeding capacities (e.g. allocating more channels than the device supports).

Some functions may not be successful due to transient conditions in the device. For example, if a tone detection message is requested, but the buffer is empty, an error will be returned. These return values will be indicated in a Return Value section of a function description.

The description for all other return values can be found in the oct6100_errors.h file of the API release.

Return values within the 0xDE000-0xDFFFF range indicate that the API software has detected an internal fatal error. These errors should be reported to Octasic for resolution.

4.2.2 Code Header Files

The user must supply C code to the API for OS and hardware specific functions. These functions are described in the User Supplied Functions Description section. The definitions of the structures needed by all user-supplied functions are provided in the oct6100_apiud.h file. The file is required by the user-supplied functions for the definitions of the structures used.

5 API Functions Description

The usage of each function and their related parameters are detailed in this section.

Almost every function has a pointer to the chip's API instance structure as its first parameter. This instance structure is created by the user before the call to **Oct6100ChipOpen** and is unique to each chip being managed by the software. This structure keeps the state of an instance of a chip and is required to perform any operations on the chip. See the System Architecture section for more details.

5.1 Chip Initialization Functions

Functions described in this section relate to the global operation of the OCT6100. They allow an application to open, close and monitor a device.

5.1.1 Oct6100ChipOpen

This function uses the tOCT6100_CHIP_OPEN configuration structure provided to perform all operations necessary to configure the chip and initialize the instance structure.

Note that the **Oct6100ChipOpenDef** and **Oct6100GetInstanceSize** functions are typically called, in their respective order, before this function.

Usage

```
#include "oct6100_api.h"

UINT32 Oct6100ChipOpenDef (
            tPOCT6100_CHIP_OPEN                 f_pChipOpen );

UINT32 Oct6100ChipOpen (
            tPOCT6100_INSTANCE_API              f_pApiInstance,
            tPOCT6100_CHIP_OPEN                 f_pChipOpen );
```

Parameters

f_pApiInstance — Pointer to the chip's API instance structure. This structure will be filled in by this function call. It contains information on the current state and configuration of the chip. Once initialized by **Oct6100ChipOpen**, this structure is supplied to all subsequent function calls. The structure must be created and kept by the application software until **Oct6100ChipClose** is called.

f_pChipOpen — Pointer to an initial tOCT6100_CHIP_OPEN configuration structure. The structure's elements are defined below. The user allocates this structure.

5.1.1.1 tOCT6100_CHIP_OPEN Structure

ulUserChipId identifier

This number is carried down to the user-supplied read/write routines to distinguish which chip the API is servicing. This can be used as an array index of the chip to be serviced to retrieve the correct instance pointer. If only one chip is being serviced by the API, then this parameter can be ignored (see the **System Architecture** description in the **Overview** section).

Direction: IN Type: UINT32

Default: 0

fMultiProcessSystem TRUE / FALSE

Indicates whether the host system is a multi-process one or not. See the **System Architecture** and **API Function Descriptions** sections for the implications of a multi-process system.

Direction: IN Type: BOOL

Default: FALSE

pProcessContext pointer

In some systems the user-provided functions (read, write, serialization, time, etc) may need a context structure in order to communicate with the host OS. This pointer is passed to all user functions for such situations. However, the parameter may be ignored by the user if it is not needed.

This parameter is copied in the instance only if **fMultiProcessSystem** is set to FALSE in the tOCT6100_OPEN_CHIP structure. If the **fMultiProcessSystem** flag is set to TRUE, the **pProcessContext** pointer can be stored in a local API instance by calling the **Oct6100CreateLocalInstance** function.

Direction: IN Type: PVOID

Default: NULL

ulMaxRwAccesses 1 – 1024

The maximum number of device addresses that the API will attempt to read or write in a single call of a user read or write function (e.g. **Oct6100DriverWriteBurstApi**).

Direction: IN Type: UINT32

Default: 8

pbyImageFile pointer

Byte pointer to the image file to be loaded into the OCT6100.

Direction: IN Type: PUINT8

Default: NULL

ulImageSize 4096 - 1048576

Size of the image file, in bytes.

Direction: IN Type: UINT32

Default: 0

ulMaxChannels 1 – 672

Maximum number of channels that this chip instance will open concurrently. This field determines the amount of memory needed by the chip's instance structure of the chip to keep track of all channels, and thus affects the required size of the instance structure.

Direction: IN Type: UINT32

Default: 672

ulTailDisplacement 0 - 896

This parameter represents the offset of the echo cancellation window, in milliseconds. This type of offset is often referred to as "bulk delay". Note that the actual tail displacement value used in the chip is in 16 ms increments. For example, if the value set in **ulTailDisplacement** is 511 ms, the actual tail displacement setting will be 496 ms.

Direction: IN Type: UINT32

Default: 0

ulMaxBiDirChannels 0 – 255

Maximum number of bi-directional channels that this chip instance will open concurrently. This field determines the amount of memory needed by the chip's instance structure of the chip to keep track of all channels, and thus affects the required size of the instance structure.

The maximum value for this parameter is also limited by the number of echo cancellation channels because each bi-directional channel requires two normal echo cancellation channels.

Direction: IN Type: UINT32

Default: 0

ulMaxTsiCncts 0 – 1530

The maximum number of TSI connections that this chip instance will open concurrently. This field determines the amount of memory needed by the chip's instance structure to keep track of all TSI connections, and thus affects the required size of the instance structure.

Direction: IN Type: UINT32

Default: 0

ulMaxPlayoutBuffers 0 – 4678

The maximum number of playout buffer that can be loaded into the chip's external memory. This field determines the amount of memory needed by the chip's instance structure to keep track of all playout buffers, and thus affects the required size of the instance structure.

The maximum number is 1344 if Caller ID is enabled (**fEnableCallerId** flag of the tOCT6100_OPEN_CHIP structure set to TRUE) and 4678 if it is not.

Direction: IN Type: UINT32

Default: 0

ulMaxConfBridges 0 – 672

The maximum number of conference bridges that this chip instance will open concurrently. This field determines the amount of memory needed by the chip's instance structure to keep track of all conference bridges, and thus affects the required size of the instance structure.

Conference bridges can be tapped for lawful interception purposes. Each time the **ulTappedChannelHndl** parameter is used when adding a monitoring participant to a bridge, the API uses 1 extra conference bridge resource. Therefore, the value of **ulMaxConfBridges** includes all conference bridges that are open as well as the number of tapping participants.

Direction: IN Type: UINT32

Default: 0

ulMaxFlexibleConfParticipants 0 – 672

The maximum number of flexible conference bridge participants that this chip instance will support concurrently. This field determines the amount of memory needed by the chip's instance structure to keep track of all flexible conference bridge participants, and thus affects the required size of the instance structure. Refer to the conference bridge section in this document for an explanation on flexible conference bridges.

Direction: IN Type: UINT32

Default: 0

ulMaxPhasingTssts 0 – 16

The maximum number of phasing TSSTs that this chip instance will open concurrently. This field determines the amount of memory needed by the chip's instance structure to keep track of all phasing TSSTs, and thus affects the required size of the instance structure.

Direction: IN Type: UINT32

Default: 0

ulMaxAdpcmChannels 0 – 672

Maximum number of ADPCM channels that this chip instance will open concurrently. This field determines the amount of memory needed by the chip's instance structure to keep track of all ADPCM channels, and thus affects the required size of the instance structure.

Direction: IN Type: UINT32

Default: 0

ulMemoryType cOCT6100_MEM_TYPE_SDR
cOCT6100_MEM_TYPE_DDR

The type of RAM memory used with the chip.

Direction: IN Type: UINT32

Default: cOCT6100_MEM_TYPE_DDR

ulMemoryChipSize cOCT6100_MEMORY_CHIP_SIZE_8MB
cOCT6100_MEMORY_CHIP_SIZE_16MB
cOCT6100_MEMORY_CHIP_SIZE_32MB
cOCT6100_MEMORY_CHIP_SIZE_64MB
cOCT6100_MEMORY_CHIP_SIZE_128MB

Indicates the size of each RAM chip used by the OCT6100. A chip size of 8 Megabytes is not supported when the memory type is set to DDR.

Direction: IN Type: UINT32

Default: cOCT6100_MEMORY_CHIP_SIZE_64MB

ulNumMemoryChips 1 – 2

Indicates the number of external RAM chips present (each of size ulMemoryChipSize). When using a 32-bit wide SDR memory device, this parameter must be set to 2 and **ulMemoryChipSize** must be set to half the size of the RAM chip.

Direction: IN Type: UINT32

Default: 1

fEnableMemClkOut TRUE / FALSE

If **ulMemoryType** is set to **cOCT6100_MEMORY_TYPE_SDR** this parameter indicates whether the pins SDRAM_CLK_O[0,1] are to be driven by the chip. If DDR RAM is used it indicates whether pins DDRAM_[CK_O, NCK_O, CK_LOCAL_O] are to be driven by the chip. If set to TRUE, then the clock is to be generated internally at the frequency specified by **ulMemClkFreq**.

Direction: IN Type: BOOL

Default: TRUE

ulUpclkFreq cOCT6100_UPCLK_FREQ_33_33_MHZ

This is the frequency of upclk. The only value allowed is 33.33 MHz. Upclk is used by the chip's CPU interface and CPU registers.

Direction: IN Type: UINT32

Default: cOCT6100_UPCLK_FREQ_33_33_MHZ

ulMemClkFreq 133000000

The frequency of the memory interface, in Hz.

If **fEnableMemClkOut** is FALSE then this parameter indicates the frequency of the oscillator.

If **fEnableMemClkOut** is TRUE, then this parameter indicates the clock frequency that the chip will generate.

Direction: IN/OUT Type: UINT32

Default: 133000000 (133 MHz)

ulInterruptPolarity cOCT6100_ACTIVE_LOW_POLARITY
cOCT6100_ACTIVE_HIGH_POLARITY

Polarity and active status of the interrupt line. The line can be active high or low and is in tri-state (open collector) when not active.

Direction: IN Type: UINT32

Default: cOCT6100_ACTIVE_LOW_POLARITY

InterruptConfig structure

See tOCT6100_INTERRUPT_CONFIGURE structure.

Direction: IN Type: tOCT6100_INTERRUPT_CONFIGURE

Default: see structure description

ulMaxTdmStreams {4, 8, 16, 32}

The maximum number of H.100 streams that this chip instance will allocate timeslots on concurrently. This parameter is used to allow the chip to operate at lower clock frequencies. When less than 32 streams are specified the most significant streams are removed first. For example, if **ulMaxTdmStreams** = 8 then only streams ct_d[7:0] can be used by this chip instance.

Direction: IN Type: UINT32

Default: 32

aulTdmStreamFreqs[8] cOCT6100_TDM_STREAM_FREQ_2MHZ
cOCT6100_TDM_STREAM_FREQ_4MHZ
cOCT6100_TDM_STREAM_FREQ_8MHZ

The frequency at which the TDM data lines operate. The streams are organized into quartets. Element 0 of this array indicates the operation frequency the TDM lines ct_d[0:3], and element 7 lines ct_d[28:31].

If **fEnableFastH100Mode** is set to TRUE, this parameter is ignored because all available streams must operate at 16 MHz.

Direction: IN Type: UINT32[8]

Default: cOCT6100_TDM_STREAM_FREQ_8MHZ

fEnableFastH100Mode TRUE / FALSE

Controls the state of the H.100 fast mode. If TRUE, the TDM bus operates at 16 MHz. Note that only streams 0 to 15 are available when operating at 16 MHz.

Setting this parameter to TRUE will cause the API to ignore the stream frequency parameter **aulTdmStreamFreqs.**

Direction: IN Type: BOOL

Default: FALSE

ulTdmSampling cOCT6100_TDM_SAMPLE_AT_3_QUARTERS
cOCT6100_TDM_SAMPLE_AT_RISING_EDGE
cOCT6100_TDM_SAMPLE_AT_FALLING_EDGE

The point from which a bit is sampled from the CT_D[31:0] lines. The bit can be sampled on the rising edge of the clock, the falling edge of the clock, or on the 3/4ths of the clock cycle.

If the **fH100FastMode** flag is set to TRUE, then only the **cOCT6100_TDM_SAMPLE_AT_RISING_EDGE** mode can be used for the TDM sampling.

Direction: IN Type: UINT32

Default: cOCT6100_TDM_SAMPLE_AT_3_QUARTERS

ulSoftToneEventsBufSize 2048 – 65535

Software buffer where the API stores tone events transferred from the chip. This value is in number of events.

Direction: IN Type: UINT32

Default: 2048

fEnable2100StopEvent TRUE / FALSE

Setting this parameter to TRUE will cause events of the type **cOCT6100_TONE_STOP** to be reported for 2100Hz tones.

Direction: IN Type: BOOL

Default: FALSE

fEnableExtToneDetection TRUE / FALSE

Setting this parameter to TRUE enables the extended tone detection mode of the OCT6100 API.

This tone detection mode allows the user to perform tone detection of a specific tone on the two voice streams of a channel (RIN and SIN). Without this mode enabled, two separate tone detectors are necessary for the two voice streams.

This parameter should only be set to TRUE when not enough tone detectors are available to cover the desired tone detection configuration. Activating this mode reduces the maximum number of echo cancellation channels to 336.

Note: All tone detectors used with extended tone detection should be configured by the OCT6100 image to perform detection on the SIN port only.

Direction: IN Type: BOOL
Default: FALSE

ulSoftBufferPlayoutEventsBufSize 128 – 65535

Software buffer where the API stores buffer playout events detected from the chip. This value is in number of events. Buffer playout events are not polled if this value is left to cOCT6100_INVALID_VALUE.

Direction: IN Type: UINT32
Default: cOCT6100_INVALID_VALUE

fEnableCallerId TRUE / FALSE

Setting this parameter to TRUE enables the caller ID module of the OCT6100 API.

Direction: IN Type: BOOL
Default: FALSE

fEnableAcousticEcho TRUE / FALSE

Activates acoustic echo cancellation in the chip. If TRUE, enabling acoustic echo cancellation will be permitted.

Direction: IN Type: BOOL
Default: FALSE

ulMaxRemoteDebugSessions 0 – 256

The maximum number of remote debugging sessions that can be supported by this instance.

Direction: IN Type: UINT32
Default: 1

fEnableChannelRecording TRUE / FALSE

If TRUE, the API will configure the device to support recording of debug information on a channel.

If **ulMaxChannels** is set to 672 and this flag is set to TRUE, the API will reserve one channel for debugging purpose, leaving only 671 channels to perform echo cancellation.

Direction: IN Type: BOOL
Default: FALSE

fAllowDynamicRecording TRUE / FALSE

If TRUE, the API will allow the user application to use a channel for debugging purposes. Debugging on the debug channel is enabled/disabled using the functions **Oct6100EnableChannelRecording** and **Oct6100DisableChannelRecording**.

Using dynamic recording gives the user application access to the device's full capacity when debugging is not required.

The **fEnableChannelRecording** flag supersedes this flag.

Direction: IN Type: BOOL

Default: FALSE

fEnableProductionBist TRUE / FALSE

If this parameter is set to TRUE, the chip will enter into production BIST mode. This mode exhaustively tests the external memory of the chip. The status of the BIST can be retrieved via a call to **Oct6100ProductionBist**. Note that the **Oct6100ChipOpen** function must be called again with this flag set to FALSE to use the chip normally.

Direction: IN Type: BOOL

Default: FALSE

ulProductionBistMode cOCT6100_PRODUCTION_BIST_STANDARD
cOCT6100_PRODUCTION_BIST_SHORT

If fEnableProductionBist is set to TRUE, this parameter will specify which BIST mode to use.

The STANDARD production BIST tests every bit of the external SDRAM and returns a pass/fail indication with some debug information concerning the failed address/data.

The SHORT production BIST is much shorter. This mode does not test every bit. It writes two values to memory, 0xAA55 and 0x55AA, reading back the values each time.

Direction: IN Type: UINT32

Default: cOCT6100_PRODUCTION_BIST_STANDARD

ulNumProductionBistLoops 0x1 – 0x10

Indicates the number of times the production BIST loop should be executed by the firmware.

A production BIST loop is composed of the following steps:

1. Walking bit set to 1.
2. Walking bit set to 0.
3. Walking bit set to 1.

The following table summarizes the production BIST duration when **ulNumProductionBistLoops** is equal to 1 for typical memory configurations:

External memory size	Approximate duration Standard BIST	Approximate duration Short BIST
32 megabytes	296 seconds	25 seconds
64 megabytes	591 seconds	50 seconds
128 megabytes	1183 seconds	100 seconds

Note that the duration increases linearly with the number of loops specified.

Direction: IN Type: UINT32

Default: 1

5.1.2 Oct6100ChipClose

This function closes all channels that may still be open and then puts the chip in soft reset.

Usage

#include "oct6100_api.h"

```
UINT32 Oct6100ChipCloseDef (
        tPOCT6100_CHIP_CLOSE            f_pChipClose );

UINT32 Oct6100ChipClose (
        tPOCT6100_INSTANCE_API          f_pApiInstance,
        tPOCT6100_CHIP_CLOSE            f_pChipClose );
```

Parameters

f_pApiInstance	Pointer to the instance structure of the chip.
f_pChipClose	Pointer to a tOCT6100_CHIP_CLOSE structure. The structure's elements are defined below. The user allocates this structure.

5.1.2.1 tOCT6100_CHIP_CLOSE Structure

At present, there are no parameters for this structure.

5.1.3 Oct6100ChipGetStats

This function fills an OCT6100_CHIP_STATS structure with the current statistics for the chip. All statistics returned by this function are initialized (e.g. counters set to 0) by the **Oct6100ChipOpen** function.

Usage

```
#include "oct6100_api.h"

UINT32 Oct6100ChipGetStatsDef (
        tPOCT6100_CHIP_STATS            f_pChipStats );

UINT32 Oct6100ChipGetStats (
        tPOCT6100_INSTANCE_API          f_pApiInstance,
        tPOCT6100_CHIP_STATS            f_pChipStats );
```

Parameters

f_pApiInstance	Pointer to the instance structure of the chip.
f_pChipStats	Pointer to a tOCT6100_CHIP_STATS structure. The structure's elements are defined below. The user allocates this structure.

5.1.3.1 tOCT6100_CHIP_STATS Structure

fResetChipStats TRUE / FALSE

If TRUE, the API resets all chip statistics counters to zero.

Direction: IN Type: BOOL
Default: FALSE

ulNumberChannels 0 – 672

The number of channels currently open.

Direction: OUT Type: UINT32
Default: cOCT6100_INVALID_STAT

ulNumberTsiCncts 0 – 1530

The number of TSI connections currently open.

Direction: OUT Type: UINT32
Default: cOCT6100_INVALID_STAT

ulNumberConfBridges 0 – 672

The number of conference bridges currently open.

Direction: OUT Type: UINT32
Default: cOCT6100_INVALID_STAT

ulNumberPlayoutBuffers 0 – 1344

The number of playout buffers currently loaded in the chip's external memory.

Direction: OUT Type: UINT32
Default: cOCT6100_INVALID_STAT

ulPlayoutFreeMemSize 0 - total space in external memory for playout

The amount of external memory left, in bytes, that can be used for buffer playout. Note that this value is not necessarily a contiguous memory block.

Direction: OUT Type: UINT32

Default: cOCT6100_INVALID_STAT

ulNumberPhasingTssts 0 – 16

The number of phasing TSSTs currently open.

Direction: OUT Type: UINT32

Default: cOCT6100_INVALID_STAT

ulNumberAdpcmChannels 0 – 672

The number of ADPCM channels currently open.

Direction: OUT Type: UINT32

Default: cOCT6100_INVALID_STAT

ulH100OutOfSynchCount 32 bit counter

A count of the number of times the H.100 slave of the chip lost its framing on the H.100 bus. The count is an approximation based on the changes from inactive to active state of the corresponding interrupt register.

Direction: OUT Type: UINT32

Default: cOCT6100_INVALID_STAT

ulH100ClockABadCount 32 bit counter

A count of the number of times the H.100 clock CT_C8_A was deemed bad. The count is an approximation based on the changes from inactive to active state of the corresponding interrupt register.

Direction: OUT Type: UINT32

Default: cOCT6100_INVALID_STAT

ulH100FrameABadCount 32 bit counter

A count of the number of times the H.100 frame CT_FRAME_A was deemed bad. The count is an approximation based on the changes from inactive to active state of the corresponding interrupt register.

Direction: OUT Type: UINT32

Default: cOCT6100_INVALID_STAT

ulH100ClockBBadCount 32 bit counter

A count of the number of times the H.100 clock CT_C8_B was deemed bad. The count is an approximation based on the changes from inactive to active state of the corresponding interrupt register.

Direction: OUT Type: UINT32

Default: cOCT6100_INVALID_STAT

ulInternalReadTimeoutCount 32 bit counter

A count of the number of times that an internal read timeout error was detected. The count is an approximation based on the changes from inactive to active state of the corresponding interrupt register.

Direction: OUT Type: UINT32

Default: cOCT6100_INVALID_STAT

ulSdramRefreshTooLateCount 32 bit counter

A count of the number of times that an SDRAM refresh too late error was detected. The count is an approximation based on the changes from inactive to active state of the corresponding interrupt register.

Direction: OUT Type: UINT32

Default: cOCT6100_INVALID_STAT

ulPllJitterErrorCount 32 bit counter

A count of the number of times that a PLL jitter error was detected. The count is an approximation based on the changes from inactive to active state of the corresponding interrupt register.

Direction: OUT Type: UINT32

Default: cOCT6100_INVALID_STAT

ulOverflowToneEventsCount 32 bit counter

A count of the number of times that the hardware tone event buffer has overflowed. The count is an approximation based on the changes from inactive to active state of the corresponding interrupt register. If such overflows occur, the user should call the Interrupt Service Routine more often.

Direction: OUT Type: UINT32

Default: cOCT6100_INVALID_STAT

ulSoftOverflowToneEventsCount 32 bit counter

A count of the number of times that the software tone event buffer has overflowed. If such overflows occur, the user should retrieve tone events from the API more often, using the **Oct6100EventGetTone** function.

Direction: OUT Type: UINT32

Default: cOCT6100_INVALID_STAT

ulSoftOverflowBufferPlayoutEventsCount 32 bit counter

A count of the number of times that the software playout event buffer has overflowed. The count is based on the number of times that buffer playout events could not be copied to the software buffer because it was already full. To correct this, the user should empty the software buffer more frequently using the **Oct6100BufferPlayoutGetEvent** procedure.

Alternatively, the **ulSoftBufferPlayoutEventsBufSize** parameter may be set to a larger value.

Direction: OUT Type: UINT32

Default: cOCT6100_INVALID_STAT

5.1.4 Oct6100ChipGetImageInfo

This function fills an OCT6100_CHIP_IMAGE_INFO structure with the description of the build image loaded into the device.

Usage

#include "oct6100_api.h"

```
UINT32 Oct6100ChipGetImageInfoDef (
        tPOCT6100_CHIP_IMAGE_INFO       f_pChipImageInfo );

UINT32 Oct6100ChipGetImageInfo (
        tPOCT6100_INSTANCE_API          f_pApiInstance,
        tPOCT6100_CHIP_IMAGE_INFO       f_pChipImageInfo );
```

Parameters

f_pApiInstance	Pointer to the instance structure of the chip.
f_pChipImageInfo	Pointer to a tOCT6100_CHIP_IMAGE_INFO structure. The structure's elements are defined below. The user allocates this structure.

5.1.4.1 tOCT6100_CHIP_IMAGE_INFO Structure

szVersionNumber[1016] String

This string contains the unique image build description of the image loaded into the device.

Direction: OUT Type: UINT8 [1016]
Default: 0

ulBuildId 32-bit value

This field contains the unique build ID from the image loaded into the device.

Direction: OUT Type: UINT32
Default: cOCT6100_INVALID_VALUE

ulImageType cOCT6100_IMAGE_TYPE_WIRELINE
cOCT6100_IMAGE_TYPE_COMBINED

This field contains the type of image that has been loaded into the device.

Direction: OUT Type: UINT32
Default: cOCT6100_INVALID_VALUE

ulMaxChannels 16 - 672

This field contains the maximum number of channels supported by the image loaded into the device.

Direction: OUT Type: UINT32
Default: cOCT6100_INVALID_VALUE

ulMaxTailDisplacement 0 - 896

This field contains the maximum tail displacement supported by the image loaded into the device. A value of 0 indicates that tail displacement is not supported.

Direction: OUT Type: UINT32

Default: cOCT6100_INVALID_VALUE

fPerChannelTailDisplacement TRUE / FALSE

If TRUE, the image loaded into the device supports per channel tail displacement configuration.

Direction: OUT Type: BOOL

Default: FALSE

fPerChannelTailLength TRUE / FALSE

If TRUE, the image loaded into the device supports an independent tail length configuration per channel.

Direction: OUT Type: BOOL

Default: FALSE

ulMaxTailLength 32 – 128 ms (increment of 4 ms)

This field contains the maximum tail length supported by the image loaded into the device.

Direction: OUT Type: UINT32

Default: cOCT6100_INVALID_VALUE

fBufferPlayout TRUE/FALSE

If TRUE, the image loaded into the device supports buffer playout.

Direction: OUT Type: BOOL

Default: FALSE

fAdaptiveNoiseReduction TRUE / FALSE

If TRUE, the image loaded into the device supports adaptive noise reduction.

Direction: OUT Type: BOOL

Default: FALSE

fSoutNoiseBleaching TRUE / FALSE

If TRUE, the image loaded into the device supports Sout noise bleaching.

Direction: OUT Type: BOOL

Default: FALSE

fAnrSnrEnhancement TRUE / FALSE

If TRUE, the image loaded into the device supports configuration of the SOUT adaptive noise reduction. This parameter controls the signal to noise ratio enhancement.

Direction: OUT Type: BOOL

Default: FALSE

fAnrVoiceNoiseSegregation TRUE / FALSE

If TRUE, the image loaded into the device supports configuration of the SOUT adaptive noise reduction. This parameter controls the voice-noise segregation.

Direction: OUT Type: BOOL

Default: FALSE

fAutoLevelControl TRUE / FALSE

If TRUE, the image loaded into the device supports automatic level control.

Direction: OUT Type: BOOL

Default: FALSE

fHighLevelCompensation **TRUE / FALSE**

If TRUE, the image loaded into the device supports high level compensation.

Direction: OUT Type: BOOL

Default: FALSE

fToneDisablerVqeActiveTime TRUE / FALSE

If TRUE, the image loaded into the device supports a configurable tone disabler VQE re-activation time.

Direction: OUT Type: BOOL

Default: FALSE

fSilenceSuppression TRUE / FALSE

If TRUE, the image loaded into the device supports silence suppression.

Direction: OUT Type: BOOL

Default: FALSE

fToneRemoval TRUE / FALSE

If TRUE, the image loaded into the device supports DTMF tone removal. This feature is available on the SIN port only.

Direction: OUT Type: BOOL

Default: FALSE

fAcousticEcho TRUE / FALSE

If TRUE, the image loaded into the device supports acoustic echo cancellation.

Direction: OUT Type: BOOL

Default: FALSE

fAecTailLength TRUE / FALSE

If TRUE, the image loaded into the device supports a configurable tail length for acoustic echo cancellation.

Direction: OUT Type: BOOL

Default: FALSE

fDefaultErl TRUE / FALSE

If TRUE, the image loaded into the device supports configurable default ERL.

Direction: OUT Type: BOOL

Default: FALSE

fNonLinearityBehaviorA TRUE / FALSE

If TRUE, the image loaded into the device supports configurable non-linearity.

Direction: OUT Type: BOOL

Default: FALSE

fNonLinearityBehaviorB TRUE / FALSE

If TRUE, the image loaded into the device supports configurable non-linearity.

Direction: OUT Type: BOOL

Default: FALSE

fDoubleTalkBehavior TRUE / FALSE

If TRUE, the image loaded into the device supports configurable double talk behavior.

Direction: OUT Type: BOOL

Default: FALSE

fListenerEnhancement TRUE / FALSE

If TRUE, the image loaded into the device supports automatic and natural listener enhancement.

Direction: OUT Type: BOOL

Default: FALSE

fMusicProtection TRUE / FALSE

If TRUE, the image loaded into the device supports Octasic's Music Protection feature.

Direction: OUT Type: BOOL

Default: FALSE

fIdleCodeDetection TRUE / FALSE

If TRUE, the image loaded into the device supports the idle code detection feature.

Direction: OUT Type: BOOL

Default: TRUE

fSinLevel TRUE / FALSE

If TRUE, the image loaded into the device supports the SIN level statistics.

Direction: OUT Type: BOOL

Default: TRUE

fConferencing TRUE / FALSE

If TRUE, the image loaded into the device supports conferencing.

Direction: OUT Type: BOOL

Default: FALSE

fConferencingNoiseReduction TRUE / FALSE

If TRUE, the image loaded into the device supports conferencing noise reduction.

Direction: OUT Type: BOOL

Default: FALSE

fDominantSpeaker TRUE / FALSE

If TRUE, the image loaded into the device supports Octasic's conferencing dominant speaker feature.

Direction: OUT Type: BOOL

Default: FALSE

fAdpcm TRUE / FALSE

If TRUE, the image loaded into the device supports ADPCM compression and decompression.

Direction: OUT Type: BOOL

Default: FALSE

ulMaxPlayoutEvents 31, 127

This field contains the maximum number of buffer playout events supported by the image loaded into the device.

Direction: OUT Type: UINT32

Default: cOCT6100_INVALID_VALUE

ulDebugEventSize 32, 256

This field contains the maximum number of debug events that can be recorded in external memory. An event is generated every 512 milliseconds. Images that support 32 events can record 16 seconds of data. Images that support 256 events can record over 2 minutes of data.

Direction: OUT Type: UINT32

Default: cOCT6100_INVALID_VALUE

ulToneProfileNumber 32-bits value

This field represents the tone profile number built in the image.

Direction: OUT Type: UINT32

Default: cOCT6100_INVALID_VALUE

ulNumTonesAvailable 0 - 56

This field represents the number of tone available in the image. It also represents the number of valid entries within the **aToneInfo** array.

Direction: OUT Type: UINT32

Default: 0

aToneInfo[56] array of structure

Description of the tones supported for tone detection based on the image loaded into the device.

Direction: OUT Type: tOCT6100_CHIP_TONE_INFO[56]

Default: see structure definition

5.1.4.2 tOCT6100_CHIP_TONE_INFO Structure

aszToneName[64] String

This field contains a unique string used to identify the tone.

Direction: OUT Type: UINT8 [64]

Default: 0

ulDetectionPort cOCT6100_CHANNEL_PORT_SIN
cOCT6100_CHANNEL_PORT_ROUT
cOCT6100_CHANNEL_PORT_SOUT
cOCT6100_CHANNEL_PORT_ROUT_SOUT

Port on which this tone can be detected.

Direction: OUT Type: UINT32

Default: cOCT6100_INVALID_PORT

ulToneID 32-bits value

Unique numerical value used to identify this tone.

This is the value required when enabling or disabling tone detection on a channel. It is also the value returned upon detection of this tone in the tOCT6100_EVENT_TONE structure. Please refer to the section on tone detection for more details.

Direction: OUT Type: UINT32

Default: cOCT6100_INVALID_VALUE

5.1.5 Oct6100GetInstanceSize

This function uses the tOCT6100_CHIP_OPEN configuration structure to calculate the amount of memory required for the tOCT6100_INSTANCE_API structure of the chip. A tOCT6100_INSTANCE_API structure must be allocated and a pointer created by the user before calling the **Oct6100ChipOpen** function; the pointer must point to a block of contiguous memory which size is determined by this function.

Usage

```
#include "oct6100_api.h"

UINT32 Oct6100GetInstanceSizeDef (
            tPOCT6100_GET_INSTANCE_SIZE     f_pInstanceSize );

UINT32 Oct6100GetInstanceSize (
            tPOCT6100_CHIP_OPEN             f_pChipOpen,
            tPOCT6100_GET_INSTANCE_SIZE     f_pInstanceSize );
```

Parameters

f_pChipOpen	Pointer to an initial tOCT6100_CHIP_OPEN configuration structure. The definition of the structure is provided in Section 5 - Configuration Parameters. See **Oct6100ChipOpenDef** for a default configuration of the chip. The user allocates this structure.
f_pInstanceSize	Pointer to a tOCT6100_GET_INSTANCE_SIZE structure. The structure's elements are defined below. The user allocates this structure.

5.1.5.1 tOCT6100_GET_INSTANCE_SIZE Structure

ulApiInstanceSize

This value is returned by the function and indicates the minimum size, in bytes, of the tOCT6100_INSTANCE_API memory block that must be allocated to support the supplied configuration.

Direction: Out Type: UINT32

Default: *NOT MODIFIED*

5.1.6 Oct6100CreateLocalInstance

This function is used only if the API is run on a multi-process system. The function initializes a local API instance structure kept by each process communicating with a given shared instance. The local structure contains all portions of the API instance which are process specific (such as serialization object handles).

Two categories of processes call this function: the main process and other processes. For example, this might occur in a system with a main process performing the channel management and a secondary thread collecting statistics.

The main process performs the configuration of the chip. In this process, this function must be called after the shared portion of the API instance is allocated but before the **Oct6100ChipOpen** function is called (see the **System Architecture** and **API Function Descriptions**).

Other processes simply connect to the chip and its shared instance once the chip is configured. This function serves as the connection function and must be called before all other API function calls (see the **System Architecture** and **API Function Descriptions**).

If the host system uses a single process then this function is not necessary, and must not be called.

Usage

```
#include "oct6100_api.h"

UINT32 Oct6100CreateLocalInstance (
            tPOCT6100_CREATE_LOCAL_INSTANCE        f_pCreateLocalInst );
```

Parameters

f_pCreateLocalInst	pointer to a tOCT6100_CREATE_LOCAL_INSTANCE structure. The definitions of the structure's elements are listed below. The user allocates this structure and keeps it as long as the chip is in operation.

5.1.6.1 tOCT6100_CREATE_LOCAL_INSTANCE Structure

pApiInstShared pointer

Pointer to the shared portion of the API instance (created by the main process). This pointer will be stored within the local instance structure (**pApiInstLocal**).

Direction: IN Type: tPOCT6100_INSTANCE_API
Default: NULL

pApiInstLocal pointer

Pointer to the process-specific portion of the API instance (created by all processes on their local stack). This pointer will be used to all subsequent API function calls.

Direction: IN Type: tPOCT6100_INSTANCE_API
Default: NULL

pProcessContext pointer

In some systems the user-provided functions (read, write, serialization, time, etc) may need a context structure in order to communicate with the host OS. The API passes this pointer to all user functions allowing the user function to retrieve the correct context. This parameter may be ignored by the user if it is not needed.

Direction: IN Type: PVOID

Default: NULL

ulUserChipId identifier

This value is passed to create the unique semaphore names that are associated to a single API instance.

Direction: IN Type: UINT32

Default: 0

5.1.7 Oct6100DestroyLocalInstance

This function should only be called in a multi-process or multi-thread environment. This function will destroy any resources reserved by a call to Oct6100CreateLocalInstance.

Usage

```
#include "oct6100_api.h"

UINT32 Oct6100DestroyLocalInstanceDef (
            tPOCT6100_DESTROY_LOCAL_INSTANCE      f_pDestroyLocalInst );

UINT32 Oct6100DestroyLocalInstance (
            tPOCT6100_INSTANCE_API                f_pApiInstance,
            tPOCT6100_DESTROY_LOCAL_INSTANCE      f_pDestroyLocalInst );
```

Parameters

f_pDestroyLocalInst — pointer to a tOCT6100_DESTROY_LOCAL_INSTANCE structure. The definitions of the structure's elements are listed below. The user allocates this structure and keeps it as long as the chip is in operation.

5.1.7.1 tOCT6100_DESTROY_LOCAL_INSTANCE Structure

ulDummy — 32 bit value

The API does not use this structure member. It exists only to preserve the OCT6100 API functions format.

Direction: IN Type: UINT32

Default: 0

5.1.8 Oct6100GetHwRevision

This routine returns the hardware revision number of the OCT6100. The revision number is contained in a register of the device. This function may be called before the device is open and only requires upclk to be present on the device.

Usage

#include "oct6100_api.h"

UINT32 Oct6100GetHwRevision (
tPOCT6100_GET_HW_REVISION f_pRevision);

Parameters

f_pRevision — pointer to a tOCT6100_GET_HW_REVISION structure. The definitions of the structure's elements are listed below. The user allocates this structure.

5.1.8.1 tOCT6100_GET_HW_REVISION Structure

ulUserChipId identifier

This value is passed to the user-supplied read/write routines to distinguish which chip the API is servicing. This can be used as an array index of the chip to be serviced to retrieve the correct instance pointer. If only one chip is being serviced by the API this parameter can be ignored. See section **1.3 System Architecture**.

Direction: IN Type: UINT32
Default: cOCT6100_INVALID_CHIP_ID

pProcessContext pointer

In some systems the user-provided functions (read, write, serialization, time, etc) may need a context structure in order to communicate with the host OS. This pointer is passed to all user functions for such situations. However, the parameter may be ignored by the user if it is not needed.

Direction: IN Type: PVOID
Default: NULL

ulRevisionNum

This value is returned by the function and indicates the revision of the device.

Direction: OUT Type: UINT32
Default: cOCT6100_INVALID_VALUE

5.1.9 Oct6100ApiGetVersion

This routine returns the version of the API as a null-terminated string. The user can call this function even if the chip is not open. This function does not require the tOCT6100_INSTANCE_API structure.

Usage

```
#include "oct6100_api.h"

UINT32 Oct6100ApiGetVersionDef (
            tPOCT6100_API_GET_VERSION           f_pApiGetVersion );

UINT32 Oct6100ApiGetVersion (
            tPOCT6100_API_GET_VERSION           f_pApiGetVersion );
```

Parameters

f_pApiGetVersion — pointer to a tOCT6100_API_GET_VERSION structure. The definitions of the structure's elements are listed below. The user allocates this structure.

5.1.9.1 tOCT6100_API_GET_VERSION Structure

achApiVersion array

This character array contains the string version of the API. This string is always terminated with a null-character.

Direction: OUT Type: UINT8[64]

Default: achApiVersion [0 - 63] = '\0';

5.1.10 Oct6100FreeResources

This routine closes all open channels and removes them from any dependencies, such as conference bridges. Optionally, other function parameters allow the user to close these other resources, such as conference bridges or TSI connections.

Usage

```
#include "oct6100_api.h"

UINT32 Oct6100FreeResourcesDef (
            tPOCT6100_ FREE_RESOURCES          f_pFreeResources );

UINT32 Oct6100FreeResources (
            tPOCT6100_INSTANCE_API             f_pApiInstance,
            tPOCT6100_FREE_RESOURCES           f_pFreeResources );
```

Parameters

f_pFreeResources — pointer to a tOCT6100_FREE_RESOURCES structure. The definitions of the structure's elements are listed below. The user allocates this structure.

5.1.10.1 tOCT6100_FREE_RESOURCES Structure

fFreeTsiConnections TRUE / FALSE

If this parameter is set to TRUE, all opened TSI connections will be closed.

Direction: IN Type: BOOL
Default: FALSE

fFreeConferenceBridges TRUE / FALSE

If this parameter is set to TRUE, all opened conference bridges will be closed.

Direction: IN Type: BOOL
Default: FALSE

fFreePlayoutBuffers TRUE / FALSE

If this parameter is set to TRUE, all loaded playout buffers in external memory will be unloaded.

Direction: IN Type: BOOL
Default: FALSE

fFreePhasingTssts TRUE / FALSE

If this parameter is set to TRUE, all opened phasing TSSTs will be closed.

Direction: IN Type: BOOL
Default: FALSE

fFreeAdpcmChannels TRUE / FALSE

If this parameter is set to TRUE, all opened ADPCM channels will be closed.

Direction: IN Type: BOOL
Default: FALSE

5.1.11 Oct6100ProductionBist

This routine returns the current production BIST status information through the **tOCT6100_PRODUCTION_BIST** structure members. Typically, the user will call this function periodically until the **ulBistStatus** member changes from cOCT6100_BIST_IN_PROGRESS to another value. This function is only available if the **fEnableProductionBist** flag of the **tOCT6100_CHIP_OPEN** structure was set to TRUE.

Usage

```
#include "oct6100_api.h"

UINT32 Oct6100ProductionBistDef (
            tPOCT6100_PRODUCTION_BIST       f_pProductionBist );

UINT32 Oct6100ProductionBist (
            tPOCT6100_INSTANCE_API          f_pApiInstance,
            tPOCT6100_PRODUCTION_BIST       f_pProductionBist );
```

Parameters

f_pProductionBist — pointer to a tOCT6100_PRODUCTION_BIST structure. The definitions of the structure's elements are listed below. The user allocates this structure.

5.1.11.1 tOCT6100_PRODUCTION_BIST Structure

ulCurrentLoop 0 - **ulNumProductionBistLoops**

The current BIST loop. The upper range of this parameter is defined by the **ulNumProductionBistLoops** of the **tOCT6100_OPEN_CHIP** structure.

Direction: OUT Type: UINT32
Default: cOCT6100_INVALID_VALUE

ulCurrentAddress 32-bit value

The current address in external memory being checked.

Direction: OUT Type: UINT32
Default: cOCT6100_INVALID_VALUE

ulCurrentTest 1 – 3

The current test being executed:

1. Walking bit set to '1'.
2. Walking bit set to '0'.
3. Walking bit set to '1'.

Direction: OUT Type: UINT32
Default: cOCT6100_INVALID_VALUE

ulBistStatus cOCT6100_BIST_IN_PROGRESS
cOCT6100_BIST_CONFIGURATION_FAILED
cOCT6100_BIST_STATUS_CRC_FAILED
cOCT6100_BIST_MEMORY_FAILED
cOCT6100_BIST_SUCCESS

The current external memory BIST status. Here is a brief description of each status.

cOCT6100_BIST_IN_PROGRESS

The BIST is in progress and no errors have been detected yet. **ulCurrentLoop** and **ulCurrentAddress** give an approximation of the progress.

cOCT6100_BIST_CONFIGURATION_FAILED

The initial configuration of the internal processors failed. The BIST could not take place.

cOCT6100_BIST_STATUS_CRC_FAILED

The current status event's CRC did not match the computed value. The small region in external memory used to exchange information between the API and the firmware is corrupted.

cOCT6100_BIST_MEMORY_FAILED

The external memory BIST failed at location **ulFailedAddress**. The firmware read value **ulReadValue** while expecting **ulExpectedValue**.

cOCT6100_BIST_SUCCESS

The BIST completed successfully. No errors were detected.

Direction: OUT Type: UINT32

Default: cOCT6100_BIST_IN_PROGRESS

ulFailedAddress 0x08000000 – (0x08000000 + external memory size)
cOCT6100_INVALID_VALUE

If **ulBistStatus** is set to cOCT6100_BIST_MEMORY_FAILED, this parameter represents the address in external memory where the failure occurred.

Direction: OUT Type: UINT32

Default: cOCT6100_INVALID_VALUE

ulReadValue 32-bit value

If **ulBistStatus** is set to cOCT6100_BIST_MEMORY_FAILED, this parameter represents the value read at the failed location.

Direction: OUT Type: UINT32

Default: cOCT6100_INVALID_VALUE

ulExpectedValue 32-bit value

If **ulBistStatus** is set to cOCT6100_BIST_MEMORY_FAILED, this parameter represents the expected value that should have been read at the failed location.

Direction: OUT Type: UINT32

Default: cOCT6100_INVALID_VALUE

5.1.12 Oct6100ApiGetCapacityPins

This routine returns the channel capacity supported by this chip. The API determines the capacity by reading the 4 CAPACITY pins on the device to identify their state, either '0' or '1'. If the hardware implementation incorrectly pulls these 4 pins to the wrong value, the API will return this incorrect value.

The user MUST call this function when the chip is NOT open and when no other process is using the chip. This API function is not serialized.

Usage

```
#include "oct6100_api.h"

UINT32 Oct6100ApiGetCapacityPinsDef (
            tPOCT6100_API_GET_CAPACITY_PINS     f_pGetCapacityPins );

UINT32 Oct6100ApiGetCapacityPins (
            tPOCT6100_INSTANCE_API              f_pApiInstance,
            tPOCT6100_API_GET_CAPACITY_PINS     f_pGetCapacityPins );
```

Parameters

f_pGetCapacityPins — pointer to a tPOCT6100_API_GET_CAPACITY_PINS structure. The definitions of the structure's elements are listed below. The user allocates this structure.

5.1.12.1 tOCT6100_API_GET_CAPACITY_PINS Structure

pProcessContext pointer

In some systems the user-provided functions (read, write, serialization, time, etc) may need a context structure in order to communicate with the host OS. The API passes this pointer to all user functions allowing the user function to retrieve the correct context. This parameter may be ignored by the user if it is not needed.

Direction: IN Type: PVOID
Default: NULL

ulUserChipId identifier

This value is passed to create the unique semaphore names that are associated to a single API instance.

Direction: IN Type: UINT32
Default: 0

ulMemoryType cOCT6100_MEM_TYPE_SDR
cOCT6100_MEM_TYPE_DDR

The type of RAM memory used with the chip.

Direction: IN Type: UINT32
Default: cOCT6100_MEM_TYPE_DDR

fEnableMemClkOut TRUE / FALSE

If **ulMemoryType** is set to **cOCT6100_MEMORY_TYPE_SDR** this parameter indicates whether the pins SDRAM_CLK_O[0,1] are to be driven by the chip. If DDR RAM is used it indicates whether pins DDRAM_[CK_O, NCK_O, CK_LOCAL_O] are to be driven by the chip. If set to TRUE, then the clock is to be generated internally at the frequency specified by **ulMemClkFreq**.

Direction: IN Type: BOOL

Default: TRUE

ulMemClkFreq 133000000

The frequency of the memory interface, in Hz.

If **fEnableMemClkOut** is FALSE then this parameter indicates the frequency of the oscillator.

If **fEnableMemClkOut** is TRUE, then this parameter indicates the clock frequency that the chip will generate.

Direction: IN/OUT Type: UINT32

Default: 133000000 (133 MHz)

ulCapacityValue 16,32,64,128,256,512,672

The maximum number of channels supported by this chip.

Direction: OUT Type: UINT32

Default: cOCT6100_INVALID_VALUE

5.2 Channel Functions

These functions are used to open, close and monitor echo channels.

Here is a short list of the features supported by a channel:

- *ADPCM Compression / decompression*
- *Silence suppression*
- *Adaptive noise reduction*
- *DC offset removal*
- *RIN/SOUT level control*

5.2.1 Oct6100ChannelOpen

This function opens an echo cancellation channel.

The channel configuration is broken down into 4 configuration sections: the main channel, the TDM, the VQE and the CODEC configuration sections.

If opened with all the default parameters, the channel's echo cancellation operation mode is set to power-down. The echo cancellation process must remain in this mode until the input ports (RIN and SIN) are assigned a valid H.100 TDM timeslot. Note that traffic can still go through the channel if the echo cancellation operation mode is in power-down but VQE features are not available.

Assignment of a timeslot can be done with the **Oct6100ChannelOpen** or **Oct6100ChannelModify** functions. Calling **Oct6100ChannelModify** will activate the echo cancellation process of a channel if **ulEchoOperationMode** is set to cOCT6100_ECHO_OP_MODE_NORMAL and the two input ports are assigned to a valid H.100 TDM timeslot.

This function returns a handle by which the API identifies this channel.

Usage

```
#include "oct6100_api.h"

UINT32 Oct6100ChannelOpenDef (
            tPOCT6100_CHANNEL_OPEN          f_pChannelOpen );

UINT32 Oct6100ChannelOpen (
            tPOCT6100_INSTANCE_API          f_pApiInstance,
            tPOCT6100_CHANNEL_OPEN          f_pChannelOpen );
```

Parameters

f_pApiInstance	Pointer to an instance structure of the chip.
f_pChannelOpen	Pointer to a tOCT6100_CHANNEL_OPEN structure. The structure's elements are defined below. The user allocates this structure.

5.2.1.1 tOCT6100_CHANNEL_OPEN Structure

pulChannelHndl handle

The parameter returns the handle for the created channel. This handle is a unique value that identifies the channel in all future function calls that affects this channel. The user allocates the memory for this pointer.

Direction: IN/OUT Type: PUINT32

Default: NULL

ulUserChanId 32-bit value

User specified field stored in the API channel structure. This parameter is returned with the channel handle when an event is detected for the current channel.

Direction: IN Type: UINT32

Default: cOCT6100_INVALID_VALUE

ulEchoOperationMode

cOCT6100_ECHO_OP_MODE_NORMAL
cOCT6100_ECHO_OP_MODE_HT_FREEZE
cOCT6100_ECHO_OP_MODE_HT_RESET
cOCT6100_ECHO_OP_MODE_POWER_DOWN
cOCT6100_ECHO_OP_MODE_NO_ECHO
cOCT6100_ECHO_OP_MODE_SPEECH_RECOGNITION

This parameter indicates the echo channel operation mode.

To bypass the echo canceller this parameter must be set to cOCT6100_ECHO_OP_MODE_POWER_DOWN. Use this mode for BERT testing or TDM bypass tests. The state of the channel should be set to cOCT6100_ECHO_OP_MODE_POWER_DOWN and then set to cOCT6100_ECHO_OP_MODE_NORMAL at the start of each call. This will reset the AF and NLP context. Resetting the echo-path model at the beginning of the call will ensure that the OCT6100 converges as quickly as possible on the new echo-path.

Setting the mode to cOCT6100_ECHO_OP_MODE_HT_FREEZE prevents the AF from updating its echo-path model. The OCT6100 will keep the last echo-path model found and apply it to the signal. This mode is typically only used for validation tests such as G.168 tests and is not used in applications.

Setting the mode to cOCT6100_ECHO_OP_MODE_HT_RESET clears the echo-path model, making the Adaptive Filter transparent. To render the OCT6100 completely transparent, the NLP must also be disabled. This mode is typically used for tests; controlling the AF is required for G.168 testing.

If using the cOCT6100_ECHO_OP_MODE_NO_ECHO operation mode, the **fEnableNlp** parameter of the voice quality enhancement configuration structure must be set to TRUE. This mode allows voice quality features (adaptive noise reduction, automatic level control, buffer playout, tone detection, etc…) to be used without performing echo cancellation.

Finally, the cOCT6100_ECHO_OP_MODE_SPEECH_RECOGNITION operation mode is used when echo cancellation needs to be enabled but not the NLP, while still allowing voice quality features. For this mode to work correctly, the **fEnableNlp** parameter of the voice quality enhancement configuration structure also needs to be set to TRUE, even though the NLP will not act on the signal. Also, the **ulComfortNoiseMode** must not be set to cOCT6100_COMFORT_NOISE_OFF.

Certain features can only be enabled in certain operation modes. Refer to the **Echo Operation Mode** section at the end of this document for a detailed table.

Direction: IN Type: UINT32

Default: cOCT6100_ECHO_OP_MODE_POWER_DOWN

fEnableToneDisabler TRUE / FALSE

If TRUE, echo cancellation and the NLP will be disabled on the channel upon detection of a 2100 Hz signal with phase reversals, and only the NLP will be disabled on the channel upon detection of a 2100 Hz signal without phase reversals. Echo cancellation on the channel will resume upon detection of a guard-band following the disabling signal.

Direction: IN Type: BOOL

Default: FALSE

fEnableExtToneDetection TRUE / FALSE

Setting this parameter to TRUE enables the extended tone detection mode for this channel. This mode allows detection of tones on both RIN and SIN, but at half the channel capacity.

To activate this mode, the API must be configured to support extended tone detection (done by setting **fEnableExtToneDetection** (tOCT6100_CHIP_OPEN) to TRUE). Note that the tone profile used should be one with all tones detected on the SIN port. Enabling this mode will then also perform tone detection on the RIN port.

Direction: IN Type: BOOL

Default: FALSE

TdmConfig structure

This structure contains all parameters related to the TDM interface of a channel. The RIN, ROUT, SIN and SOUT port values are assigned within this structure.

Direction: IN Type: tOCT6100_CHANNEL_OPEN_TDM

Default: see structure description

VqeConfig structure

This structure contains all the voice quality enhancement parameters.

Direction: IN Type: tOCT6100_CHANNEL_OPEN_VQE

Default: see structure description

CodecConfig structure

This structure contains all encoder/decoder related parameters.

Direction: IN Type: tOCT6100_CHANNEL_OPEN_CODEC

Default: see structure description

5.2.1.2 tOCT6100_CHANNEL_OPEN_TDM Structure

ulSinPcmLaw cOCT6100_PCM_U_LAW
cOCT6100_PCM_A_LAW

This parameter represents the PCM law of the samples read from the SIN port of the channel.

Direction: IN Type: UINT32

Default: cOCT6100_PCM_U_LAW

ulSinNumTssts 1, 2

This parameter indicates the number of TSSTs used for the SIN port. See the **TSST Formats section** for more information.

Direction: IN Type: UINT32

Default: 1

ulSinTimeslot 0 – 255 for 16 MHz stream frequency
0 – 127 for 8 MHz stream frequency
0 – 63 for 4 MHz stream frequency
0 – 31 for 2 MHz stream frequency
cOCT6100_UNASSIGNED

The TDM timeslot of the channel's SIN port. Note that allowed values are affected by the frequency of the clock that controls the **ulSinStream**.

If the value of **ulSinTimeslot** is unknown when the channel is opened, this parameter must be set to cOCT6100_UNASSIGNED. Note that if **ulSinTimeslot** is set to cOCT6100_UNASSIGNED, **ulSinStream** must also be set to cOCT6100_UNASSIGNED. This parameter can be configured later by a call to **Oct6100ChannelModify**.

Direction: IN Type: UINT32

Default: cOCT6100_UNASSIGNED

ulSinStream 0 – 31 for **ulMaxTdmStreams** of 32
0 – 15 for **ulMaxTdmStreams** of 16
0 – 7 for **ulMaxTdmStreams** of 8
0 – 3 for **ulMaxTdmStreams** of 4
cOCT6100_UNASSIGNED

The TDM stream of the channel's SIN port. Note that allowed values are affected by the **ulMaxTdmStreams** value specified at the **Oct6100ChipOpen** call.

If the value of **ulSinStream** is unknown when the channel is opened, this parameter must be set to cOCT6100_UNASSIGNED. Note that if **ulSinStream** is set to cOCT6100_UNASSIGNED, **ulSinTimeslot** must also be set to cOCT6100_UNASSIGNED. This parameter can be configured later by a call to **Oct6100ChannelModify**.

Direction: IN Type: UINT32

Default: cOCT6100_UNASSIGNED

ulRinPcmLaw cOCT6100_PCM_U_LAW
cOCT6100_PCM_A_LAW

This parameter represents the PCM law of the samples read from the RIN port of the channel.

Direction: IN Type: UINT32

Default: cOCT6100_PCM_U_LAW

ulRinNumTssts 1, 2

This parameter indicates the number of TSSTs used for the RIN port. Refer to the **TSST Formats section** for more information.

Direction: IN Type: UINT32

Default: 1

ulRinTimeslot see **ulSinTimeslot** parameter

The TDM timeslot of the channel's RIN port. Note that allowed values are affected by the frequency of the clock that controls the **ulRinStream**.

If the value of **ulRinTimeslot** is unknown when the channel is opened, this parameter must be set to cOCT6100_UNASSIGNED. Note that if **ulRinTimeslot** is set to cOCT6100_UNASSIGNED, **ulRinStream** must also be set to cOCT6100_UNASSIGNED. This parameter can be configured later by a call to **Oct6100ChannelModify**.

Direction: IN Type: UINT32

Default: cOCT6100_UNASSIGNED

ulRinStream see **ulSinStream** parameter

The TDM stream of the channel's RIN port. Note that allowed values are also affected by the **ulMaxTdmStreams** value specified at the **Oct6100ChipOpen** call.

If the value of **ulRinStream** is unknown when the channel is opened, this parameter must be set to cOCT6100_UNASSIGNED. Note that if **ulRinStream** is set to cOCT6100_UNASSIGNED, **ulRinTimeslot** must also be set to cOCT6100_UNASSIGNED. This parameter can be configured later by a call to **Oct6100ChannelModify**.

Direction: IN Type: UINT32

Default: cOCT6100_UNASSIGNED

ulSoutPcmLaw cOCT6100_PCM_U_LAW
cOCT6100_PCM_A_LAW

This parameter represents the PCM law of the samples driven by the SOUT port of the channel.

Direction: IN Type: UINT32

Default: cOCT6100_PCM_U_LAW

ulSoutNumTssts 1, 2

This parameter indicates the number of TSSTs used for the SOUT port. Refer to the **TSST Formats** section for more information.

Direction: IN Type: UINT32

Default: 1

ulSoutTimeslot see **ulSinTimeslot** parameter

The TDM timeslot of the channel's SOUT port. Note that allowed values are affected by the frequency of the clock that controls the **ulSoutStream**.

If the value of **ulSoutTimeslot** is unknown when the channel is opened, this parameter must be set to cOCT6100_UNASSIGNED. Note that if **ulSoutTimeslot** is set to cOCT6100_UNASSIGNED, **ulSoutStream** must also be set to cOCT6100_UNASSIGNED. This parameter can be configured later by a call to **Oct6100ChannelModify**.

Direction: IN Type: UINT32

Default: cOCT6100_UNASSIGNED

ulSoutStream see **ulSinStream** parameter

The TDM stream of the channel's SOUT port. Note that allowed values are also affected by the **ulMaxTdmStreams** value specified at the **Oct6100ChipOpen** call.

If the value of **ulSoutStream** is unknown when the channel is opened, this parameter must be set to cOCT6100_UNASSIGNED. Note that if **ulSoutStream** is set to cOCT6100_UNASSIGNED, **ulSoutTimeslot** must also be set to cOCT6100_UNASSIGNED. This parameter can be configured later by a call to **Oct6100ChannelModify**.

Direction: IN Type: UINT32

Default: cOCT6100_UNASSIGNED

ulRoutPcmLaw cOCT6100_PCM_U_LAW
cOCT6100_PCM_A_LAW

This parameter represents the PCM law of the samples driven by the ROUT port of the channel.

Direction: IN Type: UINT32

Default: cOCT6100_PCM_U_LAW

ulRoutNumTssts 1, 2

This parameter indicates the number of TSSTs used for the Rout port. Refer to the **TSST Formats** section for more information.

Direction: IN Type: UINT32

Default: 1

ulRoutTimeslot see **ulSinTimeslot** parameter

The TDM timeslot of the channel's ROUT port. Note that allowed values are affected by the frequency of the clock that controls the **ulRoutStream**.

If the value of **ulRoutTimeslot** is unknown when the channel is opened, this parameter must be set to cOCT6100_UNASSIGNED. Note that if **ulRoutTimeslot** is set to cOCT6100_UNASSIGNED, **ulRoutStream** must also be set to cOCT6100_UNASSIGNED. This parameter can be configured later by a call to **Oct6100ChannelModify**.

Direction: IN Type: UINT32

Default: cOCT6100_UNASSIGNED

ulRoutStream see **ulSinStream** parameter

The TDM stream of the channel's ROUT port. Note that allowed values are also affected by the **ulMaxTdmStreams** value specified at the **Oct6100ChipOpen** call.

If the value of **ulRoutStream** is unknown when the channel is opened, this parameter must be set to cOCT6100_UNASSIGNED. Note that if **ulRoutStream** is set to cOCT6100_UNASSIGNED, **ulRoutTimeslot** must also be set to cOCT6100_UNASSIGNED. This parameter can be configured later by a call to **Oct6100ChannelModify**.

Direction: IN Type: UINT32

Default: cOCT6100_UNASSIGNED

5.2.1.3 tOCT6100_CHANNEL_OPEN_VQE Structure

fEnableNlp TRUE / FALSE

If TRUE, the NLP will be activated on this channel. The NLP should only be disabled when performing controlled tests such as G.168 compliance testing or TDM bypass tests.

Some features cannot be used when this parameter is set to FALSE. Refer to **Echo operation mode** section at the end of this document for more information.

Direction: IN Type: BOOL

Default: TRUE

fEnableTailDisplacement TRUE / FALSE

If TRUE, tail displacement will be supported by this channel. The **ulTailDisplacement** parameter specifies the offset of the echo cancellation window.

Tail displacement can be used when the echo path delay exceeds the tail length. This is necessary when using a link with a fixed network delay in the echo path.

Direction: IN Type: BOOL

Default: FALSE

ulTailDisplacement 0 - 896
cOCT6100_AUTO_SELECT_TAIL

This parameter represents the offset of the echo cancellation window, in milliseconds. Setting this parameter to cOCT6100_AUTO_SELECT_TAIL uses the tail displacement value specified when the chip was opened. This parameter is ignored if **fEnableTailDisplacement** is set to FALSE. Note that the actual tail displacement value used in the chip is in 16 ms increments. For example, if the value set in **ulTailDisplacement** is 511 ms, the actual tail displacement setting will be 496 ms.

Direction: IN Type: UINT32

Default: cOCT6100_AUTO_SELECT_TAIL

ulTailLength 32 – 128 ms (increment of 4 ms)
cOCT6100_AUTO_SELECT_TAIL

This parameter represents the maximum tail length, in milliseconds, to be used on the channel. The value is specified in increments of 4 milliseconds. This value cannot be modified once the channel is opened. Setting this parameter to cOCT6100_AUTO_SELECT_TAIL uses the maximum tail length supported by the image.

Direction: IN Type: UINT32

Default: cOCT6100_AUTO_SELECT_TAIL

fSinDcOffsetRemoval TRUE / FALSE

If TRUE, the DC Offset Removal module will remove the DC offset of the SIN signal. Enabling DC offset removal will improve the performance of the echo canceller and should always be left on. It can be turned off for bypass tests.

DC offsets are introduced by various low-frequency noise sources, such as electrical interference.

Direction: IN Type: BOOL

Default: TRUE

fRinDcOffsetRemoval TRUE / FALSE

If TRUE, the DC Offset Removal module will remove the DC offset of the RIN signal.

Direction: IN Type: BOOL

Default: TRUE

fRinLevelControl TRUE / FALSE

If FALSE, the Level Control module will be bypassed. Note that the **fRinAutomaticLevelControl** flag must be set to FALSE for the Level Control module to perform correctly.

Direction: IN Type: BOOL

Default: FALSE

lRinLevelControlGainDb -24 – 24

If **fRinLevelControl** is set to TRUE, then this parameter is the gain applied to the RIN signal.

Direction: IN Type: INT32

Default: 0

fSoutLevelControl TRUE / FALSE

If FALSE, the Level Control module will be bypassed. Note that the **fSoutAutomaticLevelControl** flag must be set to FALSE for the Level Control module to perform correctly.

Direction: IN Type: BOOL

Default: FALSE

lSoutLevelControlGainDb -24 – 24

If **fSoutLevelControl** is set to TRUE, then this parameter is the gain applied to the SOUT signal.

Direction: IN Type: INT32

Default: 0

fRinAutomaticLevelControl TRUE / FALSE

When set to TRUE, the Automatic Level Control (ALC) module will be activated on the RIN path. ALC allows the user to specify a target level for the voice of the talker on the line. The level is set using the **lRinAutomaticLevelControlTargetDb** parameter. Note that the **fRinLevelControl** and **fRinHighLevelCompensation** flags must be set to FALSE for the Automatic Level Control module to perform correctly.

Direction: IN Type: BOOL

Default: FALSE

lRinAutomaticLevelControlTargetDb -40 – 0

If **fRinAutomaticLevelControl** is set to TRUE, then this parameter is the target level, in dBm0, to be reached on the RIN signal.

Direction: IN Type: INT32

Default: -20

fSoutAutomaticLevelControl TRUE / FALSE

When set to TRUE, the Automatic Level Control (ALC) module will be activated on the SOUT path. ALC allows the user to specify a target level for the voice of the talker on the line. The level is set using the **lSoutAutomaticLevelControlTargetDb** parameter. Note that the **fSoutLevelControl** flag must be set to FALSE for the Automatic Level Control module to perform correctly.

Direction: IN Type: BOOL

Default: FALSE

lSoutAutomaticLevelControlTargetDb -40 – 0

If **fSoutAutomaticLevelControl** is set to TRUE, then this parameter is the target level, in dBm0, to be reached on the SOUT signal.

Direction: IN Type: INT32

Default: -20

fRinHighLevelCompensation TRUE / FALSE

When set to TRUE, the High Level Compensation (HLC) module will be activated on the RIN path. HLC reduces the signal level when it approaches saturation, to avoid the introduction of non-linearities into the echo path. The threshold for saturation is set using the **lRinHighLevelCompensationThresholdDb** parameter. Note that the **fRinLevelControl** and **fRinAutomaticLevelControl** flags must be set to FALSE for the High Level Compensation module to perform correctly.

Direction: IN Type: BOOL

Default: FALSE

lRinHighLevelCompensationThresholdDb -40 – 0

If **fRinHighLevelCompensation** is set to TRUE, then this parameter represents, in dBm0, the automatically adjusted threshold on the RIN signal.

Direction: IN Type: INT32

Default: -10

fSoutAdaptiveNoiseReduction TRUE / FALSE

If FALSE, the Adaptive Noise Reduction module is bypassed. This feature is available only with OCT61x6 devices. If this feature is enabled in an OCT61x2 or OCT61x4 device, the function will return the error cOCT6100_ERR_NOT_SUPPORTED_CHANNEL_ANR.

Direction: IN Type: BOOL

Default: FALSE

ulComfortNoiseMode cOCT6100_COMFORT_NOISE_NORMAL
cOCT6100_COMFORT_NOISE_FAST_LATCH
cOCT6100_COMFORT_NOISE_EXTENDED
cOCT6100_COMFORT_NOISE_OFF

This parameter represents the comfort noise applied to the channel. Here is a brief description of each mode.

cOCT6100_COMFORT_NOISE_NORMAL

This mode gives optimal subjective results. It may not pass certain objective tests.

cOCT6100_COMFORT_NOISE_FAST_LATCH

This mode is similar to the normal mode but delivers rapid latching of background noise at the beginning of a call. Certain wireless carriers prefer this mode of operation. It may not pass certain objective tests.

cOCT6100_COMFORT_NOISE_EXTENDED

This mode is fully G.168 compliant. It delivers good subjective quality.

cOCT6100_COMFORT_NOISE_OFF

This mode turns off the comfort noise generation. This may be required in certain machine-connected applications such as speech recognition or power measurements.

Direction: IN Type: UINT32

Default: cOCT6100_COMFORT_NOISE_NORMAL

fDtmfToneRemoval TRUE / FALSE

If TRUE, the OCT6100 will remove any DTMF tones detected on the SIN port.

Direction: IN Type: BOOL

Default: FALSE

fAcousticEcho TRUE / FALSE

If TRUE, acoustic echo cancellation will be performed on the channel. This parameter should only be enabled in acoustic echo situations, where the echo path is highly non-linear. Enabling this parameter will lower the amount of residual echo perceived, while increasing the risk of double-talk clipping.

Direction: IN Type: BOOL

Default: FALSE

fSoutNoiseBleaching TRUE / FALSE

If TRUE, the SOUT noise bleaching module is enabled. This parameter activates a noise reduction algorithm that completely removes all background noise present while silence is detected. The target application of this algorithm is for pre-processing of the signal before mixing with another signal (such as music). Note that using this algorithm alone (without post-mixing) will not provide good subjective quality because it removes all ambient sound during silence periods.

This feature is only available with OCT61x6 devices. If this feature is enabled in an OCT61x2 or OCT61x4 device, the function will return the error cOCT6100_ERR_NOT_SUPPORTED_CHANNEL_NOISE_BLEACHING.

Direction: IN Type: BOOL

Default: FALSE

fSoutConferencingNoiseReduction TRUE / FALSE

If FALSE, the conferencing noise reduction module is bypassed. This feature is available only with OCT61x6 devices. If this feature is enabled in an OCT61x2 or OCT61x4 device, the function will return the error cOCT6100_ERR_NOT_SUPPORTED_CHANNEL_CNR.

Direction: IN Type: BOOL

Default: FALSE

ulNonLinearityBehaviorA 0 - 13

This parameter adjusts the behavior of the echo canceller when non-linear white noise affects the echo-cancelled path.

Here is a table describing the behavior of the algorithm based on the selected value.

Parameter value	Risk of double talk clipping	Risk of residual echo
0	Decreased	Increased
13	Increased	Decreased

Direction: IN Type: UINT32

Default: 1

ulNonLinearityBehaviorB 0 - 8

This parameter adjusts the behavior of the echo canceller when spectrally rich noise affects the echo-cancelled path.

Here is a table describing the behavior of the algorithm based on the selected value.

Parameter value	Risk of double talk clipping	Risk of residual echo
0	Decreased	Increased
8	Increased	Decreased

Direction: IN Type: UINT32

Default: 0

ulDoubleTalkBehavior cOCT6100_DOUBLE_TALK_BEH_NORMAL
cOCT6100_DOUBLE_TALK_BEH_LESS_AGGRESSIVE

This parameter configures the behavior of the algorithm on double talk. Setting this parameter to cOCT6100_DOUBLE_TALK_BEH_NORMAL will give the best optimal subjective results. When setting this parameter to cOCT6100_DOUBLE_TALK_BEH_LESS_AGGRESSIVE, the NLP will be less aggressive, resulting in slightly improved double talk behavior, but may leave residual echo.

The parameter cOCT6100_DOUBLE_TALK_BEH_LESS_AGGRESSIVE should only be used in network situations where the echo response is known to be very linear.

Direction: IN Type: UINT32

Default: cOCT6100_DOUBLE_TALK_BEH_NORMAL

lDefaultErlDb 0, -3, -6, -9 or -12 dB

The default ERL that is assumed by the NLP when not converged, e.g. at the beginning of a call.

Direction: IN Type: INT32

Default: -6 dB

lAecDefaultErlDb 0, -3 or -6 dB

The acoustic echo cancellation default ERL that is assumed.

Direction: IN Type: INT32

Default: 0 dB

ulAecTailLength 128, 256, 512 or 1024 ms

This parameter represents the maximum tail length, in milliseconds, to be used on the channel, when performing acoustic echo cancellation. This value must be greater than the configured channel tail length (**ulTailLength**) plus the requested tail displacement (**ulTailDisplacement**).

Direction: IN Type: UINT32

Default: 128 ms

ulSoutAutomaticListenerEnhancementGainDb 0 - 30

The SOUT automatic listener enhancement gain that will be applied. This feature adjusts the level of the Sin Path to compensate for loud noise present in the environment of the listener. This value determines by how many dBs the user wants the SIN voice to be above the RIN noise. Setting this parameter to 0 will disable this feature.

Direction: IN Type: UINT32

Default: 0

fSoutNaturalListenerEnhancement TRUE / FALSE

If TRUE, the SOUT natural listener enhancement module will be activated. The natural listener enhancement algorithm will adjust the voice level of the SOUT talker to the RIN voice level or to the value above the noise set by ulSoutAutomaticListenerEnhancementGainDb. The algorithm will use as a target the highest of the two.

Direction: IN Type: BOOL

Default: FALSE

ulSoutNaturalListenerEnhancementGainDb 0 - 30

The SOUT natural listener enhancement gain that will be applied. This value determines by how many dBs the user wants the SIN voice to be above the RIN background noise.

Direction: IN Type: UINT32

Default: 0

lAnrSnrEnhancementDb -9, -12, -15, -18, -21, -24, -27 or -30 dB

If the SOUT adaptive noise reduction module is activated, this parameter represents the attenuation that will be applied to the noise signal.

Direction: IN Type: INT32

Default: -18 dB

ulAnrVoiceNoiseSegregation 0 - 15

This parameter affects the behavior of the SOUT adaptive noise reduction algorithm. It is used to adjust the algorithm that differentiates between noise and voice. This will determine which part of the signal is reduced.

Here is a table describing the behavior of the algorithm based on the selected values.

Parameter value	Aggressiveness in considering what is a noise signal
0	Least – Only very pure background noise will be considered as background noise
6	Default - For optimal operation
15	Most – All low energy voice will be considered as background noise

Direction: IN Type: UINT32
Default: 6

ulToneDisablerVqeActivationDelay 300, 300 + N*512, 16172 ms (increments of 512 ms)

The following feature only applies if the tone disabler is activated. After detection of a 2100 Hz tone or 2100 Hz tone with phase reversal, the tone disabler disables VQE features. After the data transmission is complete, this parameter specifies the required silence period before the re-activation of the VQE features. The value is specified in increments of 512 milliseconds.

Direction: IN Type: UINT32
Default: 300

fEnableMusicProtection TRUE / FALSE

If TRUE, the Octasic Music Protection module will be activated.

Effective Music Protection means that music heard while "on hold" or in the background of speech during phone conversations is not cut, clipped or incorrectly transmitted by the echo canceller on the line.

Direction: IN Type: BOOL
Default: FALSE

fIdleCodeDetection TRUE / FALSE

If TRUE, the idle code detection module will be activated. The idle code detector will reinitialize the state of the Adaptive Filter between calls to achieve faster convergence at the beginning of each call. The idle code detector module will also reset the context of the following modules:

- Automatic Level Control
- Automatic Listener Enhancement
- High Level Compensation
- Natural Listener Enhancement

The idle code detector triggers when a low energy signal or constant DC offset is detected on RIN and SIN ports for 1 second.

Direction: IN Type: BOOL
Default: TRUE

5.2.1.4 tOCT6100_CHANNEL_OPEN_CODEC Structure

ulAdpcmNibblePosition cOCT6100_ADPCM_IN_LOW_BITS
cOCT6100_ADPCM_IN_HIGH_BITS

This is the position of the ADPCM bits within the H.100 TDM timeslot.

Direction: IN Type: UINT32

Default: cOCT6100_ADPCM_IN_LOW_BITS

ulEncoderPort cOCT6100_CHANNEL_PORT_ROUT
cOCT6100_CHANNEL_PORT_SOUT
cOCT6100_NO_ENCODING

This parameter is the channel port used by the encoder.

If set to **cOCT6100_NO_ENCODING**, no encoding can take place on this channel.

Direction: IN Type: UINT32

Default: cOCT6100_CHANNEL_PORT_SOUT

ulEncodingRate cOCT6100_G711_64KBPS
cOCT6100_G726_40KBPS
cOCT6100_G726_32KBPS
cOCT6100_G726_24KBPS
cOCT6100_G726_16KBPS
cOCT6100_G727_40KBPS_4_1
cOCT6100_G727_40KBPS_3_2
cOCT6100_G727_40KBPS_2_3
cOCT6100_G727_32KBPS_4_0
cOCT6100_G727_32KBPS_3_1
cOCT6100_G727_32KBPS_2_2
cOCT6100_G727_24KBPS_3_0
cOCT6100_G727_24KBPS_2_1
cOCT6100_G727_16KBPS_2_0

This parameter represents the rate of the encoder. G.727 defines contain a suffix: The first number is the number of core bits, and the second is the number of enhanced bits.

The API ignores this parameter if **ulEncoderPort** is set to cOCT6100_NO_ENCODING.

Direction: IN Type: UINT32

Default: cOCT6100_G711_64KBPS

ulDecoderPort cOCT6100_CHANNEL_PORT_RIN
cOCT6100_CHANNEL_PORT_SIN
cOCT6100_NO_DECODING

This parameter is the channel port used by the Decoder. The Decoder reads samples coming from the TDM interface. The samples read are decoded and then fed to the Echo Canceller module.

If set to cOCT6100_NO_DECODING, no decoding can take place on this channel.

Direction: IN Type: UINT32

Default: cOCT6100_CHANNEL_PORT_RIN

ulDecodingRate

cOCT6100_G711_64KBPS
cOCT6100_G726_40KBPS
cOCT6100_G726_32KBPS
cOCT6100_G726_24KBPS
cOCT6100_G726_16KBPS
cOCT6100_G727_2C_ENCODED
cOCT6100_G727_3C_ENCODED
cOCT6100_G727_4C_ENCODED
cOCT6100_G726_ENCODED
cOCT6100_G711_G726_ENCODED
cOCT6100_G711_G727_2C_ENCODED
cOCT6100_G711_G727_3C_ENCODED
cOCT6100_G711_G727_4C_ENCODED

This parameter represents the rate of the decoder. G.727 defines contain a suffix: The first number is the number of core bits, and the second is the number of enhanced bits

If the decoding rate is a combination of G.711 with either G.726 or G.727, the number of TSSTs assigned to the Decoder input port must be set to 2.

The API ignores this parameter if **ulDecoderPort** is set to cOCT6100_NO_DECODING.

Direction: IN Type: UINT32

Default: cOCT6100_G711_64KBPS

fEnableSilenceSuppression TRUE / FALSE

Silence suppression can be enabled only if **ulEncoderPort** is set to cOCT6100_CHANNEL_PORT_SOUT. Silence suppression is only active when the **ulEchoOperationMode** parameter is not set to cOCT6100_ECHO_OP_MODE_POWER_DOWN.

Silence suppression requires a valid phasing TSST.

The API ignores this parameter if **ulEncoderPort** is set to cOCT6100_NO_ENCODING.

Direction: IN Type: BOOL

Default: FALSE

ulPhasingType

cOCT6100_SINGLE_PHASING
cOCT6100_DUAL_PHASING
cOCT6100_NO_PHASING

It indicates how **ulPhase** is interpreted. See **ulPhase** description for more information. Setting this parameter to cOCT6100_NO_PHASING specifies that no phasing TSST is associated with this channel.

The API ignores this parameter if **ulEncoderPort** is set to cOCT6100_NO_ENCODING.
Note that silence suppression cannot be enabled (**fEnableSilenceSuppression** set to TRUE) if this parameter is set to cOCT6100_NO_PHASING. The OCT6100 device requires a valid phasing TSST to output the silence suppression information on the H.100 bus.

Direction: IN Type: UINT32

Default: cOCT6100_NO_PHASING

ulPhase 1 – **ulPhasing_length** –1

If **ulPhasingType** specifies cOCT6100_SINGLE_PHASING or cOCT6100_DUAL_PHASING, then the specific phase value used to identify the beginning of a packetization boundary must be specified. The **ulPhase** indicates the frame in which the external agent SAR is fetching the first sample used to assemble the next packet.

When cOCT6100_DUAL_PHASING is specified, the **ulPhase** indicates the beginning of an ADPCM packet boundary and must be even. A PCM packet boundary must also exist at (**ulPhase** + (**ulPhasingLength**/2)) MOD **ulPhasingLength**.

ulPhasingLength is specified when a phasing TSST is opened by the **Oct6100PhasingTsstOpen** function.

If **ulPhasingType** is set to cOCT6100_NO_PHASING, this parameter is ignored.

The API ignores this parameter if **ulEncoderPort** is set to cOCT6100_NO_ENCODING.

Direction: IN Type: UINT32

Default: 1

ulPhasingTsstHndl phasing TSST handle

If silence suppression is enabled, a valid phasing TSST handle must be specified. A phasing TSST handle is returned by the **Oct6100PhasingTsstOpen** function.

If **ulEncoderPort** is set to cOCT6100_NO_ENCODING or **ulPhasingType** is set to cOCT6100_NO_PHASING, this parameter is ignored.

Direction: IN Type: UINT32

Default: cOCT6100_INVALID_HANDLE

5.2.2 Oct6100ChannelClose

This function closes a channel.

Usage

```
#include "oct6100_api.h"

UINT32 Oct6100ChannelCloseDef (
        tPOCT6100_CHANNEL_CLOSE        f_pChannelClose );

UINT32 Oct6100ChannelClose (
        tPOCT6100_INSTANCE_API         f_pApiInstance,
        tPOCT6100_CHANNEL_CLOSE        f_pChannelClose );
```

Parameters

f_pApiInstance	Pointer to an instance structure of the chip.
f_pChannelClose	Pointer to a tOCT6100_CHANNEL_CLOSE structure. The structure's elements are defined below. The user allocates this structure.

5.2.2.1 tOCT6100_CHANNEL_CLOSE Structure

ulChannelHndl handle

This is the handle of the channel to be closed. This value was returned by a call to **Oct6100ChannelOpen**.

Direction: IN Type: UINT32
Default: cOCT6100_INVALID_HANDLE

5.2.3 Oct6100ChannelModify

This function allows the user to dynamically change some of the channel configuration parameters.

In addition to the main channel state, parameters are separated into three categories: TSST, CODEC and VQE.

Each categories is a member of the modify structure and has a modified flag within that main structure (except for the main state). The API will process the changes within a category only if the modified flag is set to TRUE.

Setting a parameter to cOCT6100_KEEP_PREVIOUS_SETTING leaves its value unchanged. Leaving all default values unchanged (except **ulChannelHndl**) will result in no modifications being performed on the channel.

Usage

```
#include "oct6100_api.h"

UINT32 Oct6100ChannelModifyDef (
            tPOCT6100_CHANNEL_MODIFY        f_pChannelModify );

UINT32 Oct6100ChannelModify (
            tPOCT6100_INSTANCE_API          f_pApiInstance,
            tPOCT6100_CHANNEL_MODIFY        f_pChannelModify );
```

Parameters

f_pApiInstance	Pointer to an instance structure of the chip.
f_pChannelModify	Pointer to a tOCT6100_CHANNEL_MODIFY structure. The structure's elements are defined below. The user allocates this structure.

5.2.3.1 tOCT6100_CHANNEL_MODIFY Structure

ulChannelHndl handle

The handle of the channel on which parameters are to be changed. This value is returned by a call to **Oct6100ChannelOpen**. This parameter will be ignored if **fApplyToAllChannels** is set to TRUE.

Direction: IN Type: UINT32

Default: cOCT6100_INVALID_HANDLE

ulUserChanId 32-bit value
cOCT6100_KEEP_PREVIOUS_SETTING

See tOCT6100_CHANNEL_OPEN Structure.

Direction: IN Type: UINT32

Default: cOCT6100_KEEP_PREVIOUS_SETTING

ulEchoOperationMode cOCT6100_ECHO_OP_MODE_NORMAL
cOCT6100_ECHO_OP_MODE_HT_FREEZE
cOCT6100_ECHO_OP_MODE_HT_RESET
cOCT6100_ECHO_OP_MODE_POWER_DOWN
cOCT6100_ECHO_OP_MODE_NO_ECHO
cOCT6100_ECHO_OP_MODE_SPEECH_RECOGNITION
cOCT6100_KEEP_PREVIOUS_SETTING

See tOCT6100_CHANNEL_OPEN Structure.

Direction: IN Type: UINT32

Default: cOCT6100_KEEP_PREVIOUS_SETTING

fEnableToneDisabler TRUE / FALSE
cOCT6100_KEEP_PREVIOUS_SETTING

See tOCT6100_CHANNEL_OPEN Structure.

Direction: IN Type: BOOL

Default: cOCT6100_KEEP_PREVIOUS_SETTING

fApplyToAllChannels TRUE / FALSE

If TRUE, the changes specified by this function call will be applied to all channels that are currently open. Note that the API semaphore will be held in locked state during the whole time it takes to modify all channels.

Direction: IN Type: BOOL

Default: FALSE

fDisableToneDetection TRUE / FALSE

If TRUE, tone detection will be disabled, for all tones currently detected on the channel.

Direction: IN Type: BOOL

Default: FALSE

fStopBufferPlayout TRUE / FALSE

If TRUE, buffer playout will be stopped as soon as possible, without necessarily reaching the end of the buffer. This will also clear the buffer playout list.

Direction: IN Type: BOOL

Default: FALSE

fRemoveConfBridgeParticipant TRUE / FALSE

If TRUE, the channel will be removed from any conference bridge that it is on. The API will not return an error if the channel is not a participant on a conference bridge.

Direction: IN Type: BOOL

Default: FALSE

fRemoveBroadcastTssts TRUE / FALSE

If TRUE, all broadcast timeslots associated to the channel will be removed. The API will not return an error if no broadcast timeslots are associated to the channel.

Direction: IN Type: BOOL

Default: FALSE

fTdmConfigModified TRUE / FALSE

This flag indicates modifications are requested in the **TdmConfig** structure. Setting this flag to FALSE will cause the API to ignore any modifications requested in the **TdmConfig** structure.

Direction: IN Type: BOOL

Default: FALSE

fVqeConfigModified TRUE / FALSE

This flag indicates modifications are requested in the **VqeConfig** structure. Setting this flag to FALSE will cause the API to ignore any modifications requested in the **VqeConfig** structure.

Direction: IN Type: BOOL

Default: FALSE

fCodecConfigModified TRUE / FALSE

This flag indicates modifications are requested in the **CodecConfig** structure. Setting this flag to FALSE will cause the API to ignore any modifications requested in the **CodecConfig** structure.

Direction: IN Type: BOOL

Default: FALSE

TdmConfig structure

This structure contains all parameters related to the TDM interface of a channel. The RIN, ROUT, SIN and SOUT port values are assigned within this structure.

Direction: IN Type: tOCT6100_CHANNEL_MODIFY_TDM

Default: see structure description

VqeConfig structure

This structure contains all the voice quality enhancement parameters.

Direction: IN Type: tOCT6100_CHANNEL_MODIFY_VQE

Default: see structure description

CodecConfig structure

This structure contains all encoder/decoder related parameters.

Direction: IN Type: tOCT6100_CHANNEL_MODIFY_CODEC

Default: see structure description

5.2.3.2 tOCT6100_CHANNEL_MODIFY_TDM Structure

ulSinPcmLaw cOCT6100_PCM_U_LAW
cOCT6100_PCM_A_LAW
cOCT6100_KEEP_PREVIOUS_SETTING

See tOCT6100_CHANNEL_OPEN Structure.

Direction: IN Type: UINT32

Default: cOCT6100_KEEP_PREVIOUS_SETTING

ulSinNumTssts 1, 2
cOCT6100_KEEP_PREVIOUS_SETTING

See tOCT6100_CHANNEL_OPEN Structure.

Direction: IN Type: UINT32

Default: cOCT6100_KEEP_PREVIOUS_SETTING

ulSinTimeslot 0 – 255 for 16 MHz stream frequency
0 – 127 for 8 MHz stream frequency
0 – 63 for 4 MHz stream frequency
0 – 31 for 2 MHz stream frequency
cOCT6100_UNASSIGNED
cOCT6100_KEEP_PREVIOUS_SETTING

See tOCT6100_CHANNEL_OPEN Structure.

If **fApplyToAllChannels** is set to TRUE, this parameter must be either cOCT6100_UNASSIGNED or cOCT6100_KEEP_PREVIOUS_SETTING.

Direction: IN Type: UINT32

Default: cOCT6100_KEEP_PREVIOUS_SETTING

ulSinStream 0 – 31 for **ulMaxTdmStreams** of 32
0 – 15 for **ulMaxTdmStreams** of 16
0 – 7 for **ulMaxTdmStreams** of 8
0 – 3 for **ulMaxTdmStreams** of 4
cOCT6100_UNASSIGNED
cOCT6100_KEEP_PREVIOUS_SETTING

See tOCT6100_CHANNEL_OPEN Structure.

If **fApplyToAllChannels** is set to TRUE, this parameter must be either cOCT6100_UNASSIGNED or cOCT6100_KEEP_PREVIOUS_SETTING.

Direction: IN Type: UINT32

Default: cOCT6100_KEEP_PREVIOUS_SETTING

ulRinPcmLaw cOCT6100_PCM_U_LAW
cOCT6100_PCM_A_LAW
cOCT6100_KEEP_PREVIOUS_SETTING

See tOCT6100_CHANNEL_OPEN Structure.

Direction: IN Type: UINT32

Default: cOCT6100_KEEP_PREVIOUS_SETTING

ulRinNumTssts 1, 2
cOCT6100_KEEP_PREVIOUS_SETTING

See tOCT6100_CHANNEL_OPEN Structure.
Direction: IN Type: UINT32
Default: cOCT6100_KEEP_PREVIOUS_SETTING

ulRinTimeslot see **ulSinTimeslot** parameter

See tOCT6100_CHANNEL_OPEN Structure.
Direction: IN Type: UINT32
Default: cOCT6100_KEEP_PREVIOUS_SETTING

ulRinStream see **ulSinStream** parameter

See tOCT6100_CHANNEL_OPEN Structure.
Direction: IN Type: UINT32
Default: cOCT6100_KEEP_PREVIOUS_SETTING

ulSoutPcmLaw cOCT6100_PCM_U_LAW
cOCT6100_PCM_A_LAW
cOCT6100_KEEP_PREVIOUS_SETTING

See tOCT6100_CHANNEL_OPEN Structure.
Direction: IN Type: UINT32
Default: cOCT6100_KEEP_PREVIOUS_SETTING

ulSoutNumTssts 1, 2
cOCT6100_KEEP_PREVIOUS_SETTING

See tOCT6100_CHANNEL_OPEN Structure.
Direction: IN Type: UINT32
Default: cOCT6100_KEEP_PREVIOUS_SETTING

ulSoutTimeslot see **ulSinTimeslot** parameter

See tOCT6100_CHANNEL_OPEN Structure.
Direction: IN Type: UINT32
Default: cOCT6100_KEEP_PREVIOUS_SETTING

ulSoutStream see **ulSinStream** parameter

See tOCT6100_CHANNEL_OPEN Structure.
Direction: IN Type: UINT32
Default: cOCT6100_KEEP_PREVIOUS_SETTING

ulRoutPcmLaw cOCT6100_PCM_U_LAW
cOCT6100_PCM_A_LAW
cOCT6100_KEEP_PREVIOUS_SETTING

See tOCT6100_CHANNEL_OPEN Structure.
Direction: IN Type: UINT32
Default: cOCT6100_KEEP_PREVIOUS_SETTING

ulRoutNumTssts 1, 2
cOCT6100_KEEP_PREVIOUS_SETTING

See tOCT6100_CHANNEL_OPEN Structure.
Direction: IN Type: UINT32
Default: cOCT6100_KEEP_PREVIOUS_SETTING

ulRoutTimeslot see **ulSinTimeslot** parameter

See tOCT6100_CHANNEL_OPEN Structure.
Direction: IN Type: UINT32
Default: cOCT6100_KEEP_PREVIOUS_SETTING

ulRoutStream see **ulSinStream** parameter

See tOCT6100_CHANNEL_OPEN Structure.
Direction: IN Type: UINT32
Default: cOCT6100_KEEP_PREVIOUS_SETTING

5.2.3.3 tOCT6100_CHANNEL_MODIFY_VQE Structure

fEnableNlp TRUE / FALSE
cOCT6100_KEEP_PREVIOUS_SETTING

See tOCT6100_CHANNEL_OPEN Structure.
Direction: IN Type: BOOL
Default: cOCT6100_KEEP_PREVIOUS_SETTING

fEnableTailDisplacement TRUE / FALSE
cOCT6100_KEEP_PREVIOUS_SETTING

See tOCT6100_CHANNEL_OPEN Structure.
Direction: IN Type: BOOL
Default: cOCT6100_KEEP_PREVIOUS_SETTING

ulTailDisplacement 0 – 896
cOCT6100_KEEP_PREVIOUS_SETTING
cOCT6100_AUTO_SELECT_TAIL

See tOCT6100_CHANNEL_OPEN Structure.
Direction: IN Type: UINT32
Default: cOCT6100_KEEP_PREVIOUS_SETTING

fSinDcOffsetRemoval TRUE / FALSE
cOCT6100_KEEP_PREVIOUS_SETTING

See tOCT6100_CHANNEL_OPEN Structure.
Direction: IN Type: BOOL
Default: cOCT6100_KEEP_PREVIOUS_SETTING

fRinDcOffsetRemoval TRUE / FALSE
cOCT6100_KEEP_PREVIOUS_SETTING

See tOCT6100_CHANNEL_OPEN Structure.
Direction: IN Type: BOOL
Default: cOCT6100_KEEP_PREVIOUS_SETTING

fRinLevelControl TRUE / FALSE
cOCT6100_KEEP_PREVIOUS_SETTING

See tOCT6100_CHANNEL_OPEN Structure.
Direction: IN Type: BOOL
Default: cOCT6100_KEEP_PREVIOUS_SETTING

lRinLevelControlGainDb -24 – 24
cOCT6100_KEEP_PREVIOUS_SETTING

See tOCT6100_CHANNEL_OPEN Structure.
Direction: IN Type: UINT32
Default: cOCT6100_KEEP_PREVIOUS_SETTING

fSoutLevelControl TRUE / FALSE
cOCT6100_KEEP_PREVIOUS_SETTING

See tOCT6100_CHANNEL_OPEN Structure.
Direction: IN Type: BOOL
Default: cOCT6100_KEEP_PREVIOUS_SETTING

lSoutLevelControlGainDb -24 – 24
cOCT6100_KEEP_PREVIOUS_SETTING

See tOCT6100_CHANNEL_OPEN Structure.
Direction: IN Type: UINT32
Default: cOCT6100_KEEP_PREVIOUS_SETTING

fRinAutomaticLevelControl TRUE / FALSE
cOCT6100_KEEP_PREVIOUS_SETTING

See tOCT6100_CHANNEL_OPEN Structure.
Direction: IN Type: BOOL
Default: cOCT6100_KEEP_PREVIOUS_SETTING

lRinAutomaticLevelControlTargetDb -40 – 0
cOCT6100_KEEP_PREVIOUS_SETTING

See tOCT6100_CHANNEL_OPEN Structure.
Direction: IN Type: INT32
Default: cOCT6100_KEEP_PREVIOUS_SETTING

fSoutAutomaticLevelControl TRUE / FALSE
cOCT6100_KEEP_PREVIOUS_SETTING

See tOCT6100_CHANNEL_OPEN Structure.
Direction: IN Type: BOOL
Default: cOCT6100_KEEP_PREVIOUS_SETTING

lSoutAutomaticLevelControlTargetDb -40 – 0
cOCT6100_KEEP_PREVIOUS_SETTING

See tOCT6100_CHANNEL_OPEN Structure.
Direction: IN Type: INT32
Default: cOCT6100_KEEP_PREVIOUS_SETTING

fRinHighLevelCompensation TRUE / FALSE
cOCT6100_KEEP_PREVIOUS_SETTING

See tOCT6100_CHANNEL_OPEN Structure.
Direction: IN Type: BOOL
Default: cOCT6100_KEEP_PREVIOUS_SETTING

lRinHighLevelCompensationThresholdDb -40 – 0
cOCT6100_KEEP_PREVIOUS_SETTING

See tOCT6100_CHANNEL_OPEN Structure.
Direction: IN Type: INT32
Default: cOCT6100_KEEP_PREVIOUS_SETTING

fSoutAdaptiveNoiseReduction TRUE / FALSE
cOCT6100_KEEP_PREVIOUS_SETTING

See tOCT6100_CHANNEL_OPEN Structure.

Direction: IN Type: BOOL

Default: cOCT6100_KEEP_PREVIOUS_SETTING

ulComfortNoiseMode cOCT6100_COMFORT_NOISE_NORMAL
cOCT6100_COMFORT_NOISE_FAST_LATCH
cOCT6100_COMFORT_NOISE_EXTENDED
cOCT6100_COMFORT_NOISE_OFF
cOCT6100_KEEP_PREVIOUS_SETTING

See tOCT6100_CHANNEL_OPEN Structure.

Direction: IN Type: UINT32

Default: cOCT6100_KEEP_PREVIOUS_SETTING

fDtmfToneRemoval TRUE / FALSE
cOCT6100_KEEP_PREVIOUS_SETTING

See tOCT6100_CHANNEL_OPEN Structure.

Direction: IN Type: BOOL

Default: cOCT6100_KEEP_PREVIOUS_SETTING

fAcousticEcho TRUE / FALSE
cOCT6100_KEEP_PREVIOUS_SETTING

See tOCT6100_CHANNEL_OPEN Structure.

Direction: IN Type: BOOL

Default: cOCT6100_KEEP_PREVIOUS_SETTING

fSoutConferencingNoiseReduction TRUE / FALSE
cOCT6100_KEEP_PREVIOUS_SETTING

See tOCT6100_CHANNEL_OPEN Structure.

Direction: IN Type: BOOL

Default: cOCT6100_KEEP_PREVIOUS_SETTING

fSoutConferencingNoiseReduction TRUE / FALSE
cOCT6100_KEEP_PREVIOUS_SETTING

See tOCT6100_CHANNEL_OPEN Structure.

Direction: IN Type: BOOL

Default: cOCT6100_KEEP_PREVIOUS_SETTING

ulNonLinearityBehaviorA 0 - 13
cOCT6100_KEEP_PREVIOUS_SETTING

See tOCT6100_CHANNEL_OPEN Structure.
Direction: IN Type: UINT32
Default: cOCT6100_KEEP_PREVIOUS_SETTING

ulNonLinearityBehaviorB 0 – 8
cOCT6100_KEEP_PREVIOUS_SETTING

See tOCT6100_CHANNEL_OPEN Structure.
Direction: IN Type: UINT32
Default: cOCT6100_KEEP_PREVIOUS_SETTING

ulDoubleTalkBehavior cOCT6100_DOUBLE_TALK_BEH_NORMAL
cOCT6100_DOUBLE_TALK_BEH_LESS_AGGRESSIVE
cOCT6100_KEEP_PREVIOUS_SETTING

See tOCT6100_CHANNEL_OPEN Structure.
Direction: IN Type: UINT32
Default: cOCT6100_KEEP_PREVIOUS_SETTING

lDefaultErlDb 0, -3, -6, -9 or -12 dB
cOCT6100_KEEP_PREVIOUS_SETTING

See tOCT6100_CHANNEL_OPEN Structure.
Direction: IN Type: INT32
Default: cOCT6100_KEEP_PREVIOUS_SETTING

lAecDefaultErlDb 0, -3 or -6 dB
cOCT6100_KEEP_PREVIOUS_SETTING

See tOCT6100_CHANNEL_OPEN Structure.
Direction: IN Type: INT32
Default: cOCT6100_KEEP_PREVIOUS_SETTING

ulAecTailLength 128, 256, 512 or 1024 ms
cOCT6100_KEEP_PREVIOUS_SETTING

See tOCT6100_CHANNEL_OPEN Structure.
Direction: IN Type: UINT32
Default: cOCT6100_KEEP_PREVIOUS_SETTING

ulSoutAutomaticListenerEnhancementGainDb 0 – 30
cOCT6100_KEEP_PREVIOUS_SETTING

See tOCT6100_CHANNEL_OPEN Structure.
Direction: IN Type: UINT32
Default: cOCT6100_KEEP_PREVIOUS_SETTING

fSoutNaturalListenerEnhancement TRUE / FALSE
cOCT6100_KEEP_PREVIOUS_SETTING

See tOCT6100_CHANNEL_OPEN Structure.
Direction: IN Type: BOOL
Default: cOCT6100_KEEP_PREVIOUS_SETTING

ulSoutNaturalListenerEnhancementGainDb 0 - 30

See tOCT6100_CHANNEL_OPEN Structure.
Direction: IN Type: UINT32
Default: 0

lAnrSnrEnhancementDb -9, -12, -15, -18, -21, -24, -27 or -30 dB
cOCT6100_KEEP_PREVIOUS_SETTING

See tOCT6100_CHANNEL_OPEN Structure.
Direction: IN Type: INT32
Default: cOCT6100_KEEP_PREVIOUS_SETTING

ulAnrVoiceNoiseSegregation 0 – 15
cOCT6100_KEEP_PREVIOUS_SETTING

See tOCT6100_CHANNEL_OPEN Structure.
Direction: IN Type: UINT32
Default: cOCT6100_KEEP_PREVIOUS_SETTING

ulToneDisablerVqeActivationDelay 300, 300 + N*512, 16172 ms
(increment of 512 ms)
cOCT6100_KEEP_PREVIOUS_SETTING

See tOCT6100_CHANNEL_OPEN Structure.
Direction: IN Type: UINT32
Default: cOCT6100_KEEP_PREVIOUS_SETTING

fEnableMusicProtection TRUE / FALSE

See tOCT6100_CHANNEL_OPEN Structure.
Direction: IN Type: BOOL
Default: cOCT6100_KEEP_PREVIOUS_SETTING

fIdleCodeDetection TRUE / FALSE

See tOCT6100_CHANNEL_OPEN Structure.
Direction: IN Type: BOOL
Default: cOCT6100_KEEP_PREVIOUS_SETTING

5.2.3.4 tOCT6100_CHANNEL_MODIFY_CODEC Structure

ulEncoderPort cOCT6100_CHANNEL_PORT_ROUT
cOCT6100_CHANNEL_PORT_SOUT
cOCT6100_NO_ENCODING
cOCT6100_KEEP_PREVIOUS_SETTING

See tOCT6100_CHANNEL_OPEN Structure.

Direction: IN Type: UINT32

Default: cOCT6100_KEEP_PREVIOUS_SETTING

ulEncodingRate cOCT6100_G711_64KBPS
cOCT6100_G726_40KBPS
cOCT6100_G726_32KBPS
cOCT6100_G726_24KBPS
cOCT6100_G726_16KBPS
cOCT6100_G727_40KBPS_4_1
cOCT6100_G727_40KBPS_3_2
cOCT6100_G727_40KBPS_2_3
cOCT6100_G727_32KBPS_4_0
cOCT6100_G727_32KBPS_3_1
cOCT6100_G727_32KBPS_2_2
cOCT6100_G727_24KBPS_3_0
cOCT6100_G727_24KBPS_2_1
cOCT6100_G727_16KBPS_2_0
cOCT6100_KEEP_PREVIOUS_SETTING

See tOCT6100_CHANNEL_OPEN Structure.

Direction: IN Type: UINT32

Default: cOCT6100_KEEP_PREVIOUS_SETTING

ulDecoderPort cOCT6100_CHANNEL_PORT_RIN
cOCT6100_CHANNEL_PORT_SIN
cOCT6100_NO_DECODING
cOCT6100_KEEP_PREVIOUS_SETTING

See tOCT6100_CHANNEL_OPEN Structure.

Direction: IN Type: UINT32

Default: cOCT6100_KEEP_PREVIOUS_SETTING

ulDecodingRate cOCT6100_G711_64KBPS
cOCT6100_G726_40KBPS
cOCT6100_G726_32KBPS
cOCT6100_G726_24KBPS
cOCT6100_G726_16KBPS
cOCT6100_G727_2C_ENCODED
cOCT6100_G727_3C_ENCODED
cOCT6100_G727_4C_ENCODED
cOCT6100_G726_ENCODED
cOCT6100_G711_G726_ENCODED
cOCT6100_G711_G727_2C_ENCODED
cOCT6100_G711_G727_3C_ENCODED
cOCT6100_G711_G727_4C_ENCODED
cOCT6100_KEEP_PREVIOUS_SETTING

See tOCT6100_CHANNEL_OPEN Structure.

Direction: IN Type: UINT32

Default: cOCT6100_KEEP_PREVIOUS_SETTING

fEnableSilenceSuppression TRUE / FALSE
cOCT6100_KEEP_PREVIOUS_SETTING

See tOCT6100_CHANNEL_OPEN Structure.

Direction: IN Type: BOOL

Default: cOCT6100_KEEP_PREVIOUS_SETTING

ulPhasingType cOCT6100_SINGLE_PHASING
cOCT6100_DUAL_PHASING
cOCT6100_NO_PHASING
cOCT6100_KEEP_PREVIOUS_SETTING

See tOCT6100_CHANNEL_OPEN Structure.

Direction: IN Type: UINT32

Default: cOCT6100_KEEP_PREVIOUS_SETTING

ulPhase 0 – **ulPhasing_length** –1
cOCT6100_KEEP_PREVIOUS_SETTING

See tOCT6100_CHANNEL_OPEN Structure.

Direction: IN Type: UINT32

Default: cOCT6100_KEEP_PREVIOUS_SETTING

ulPhasingTsstHndl phasing TSST handle
cOCT6100_KEEP_PREVIOUS_SETTING

See tOCT6100_CHANNEL_OPEN Structure.

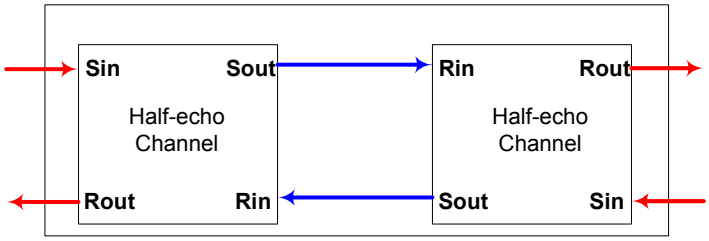
Direction: IN Type: UINT32

Default: cOCT6100_KEEP_PREVIOUS_SETTING

5.2.4 Oct6100ChannelCreateBiDir

This function creates a bi-directional channel. A bi-directional channel is composed of two echo canceller channels where the SOUT port of each channel is connected to the RIN port of the other channel, as illustrated in the following drawing:

Bi-directional echo channel



A bi-directional channel requires that only 4 ports (SIN * 2 and ROUT * 2) be connected to the TDM bus to perform echo cancellation on both echo cancellation channels.

To do this, the API binds two channels into a bi-directional channel. The channels must be properly configured for the bind operation to succeed. A channel is deemed properly configured if it supports the following configuration:

- The timeslot and stream values for the SOUT and RIN port must be set to cOCT6100_UNASSIGNED
- No PCM law translation
- No ADPCM compression or decompression
- No silence suppression
- No conferencing

After the user binds the channels together using this function, each channel can still be modified independently using the **Oct6100ChannelModify** function.

Usage

```
#include "oct6100_api.h"

UINT32 Oct6100ChannelCreateBiDirDef (
            tPOCT6100_CHANNEL_CREATE_BIDIR      f_pChannelCreateBiDir );

UINT32 Oct6100ChannelCreateBiDir (
            tPOCT6100_INSTANCE_API              f_pApiInstance,
            tPOCT6100_CHANNEL_CREATE_BIDIR      f_pChannelCreateBiDir );
```

Parameters

f_pApiInstance	Pointer to an instance structure of the chip.
f_pChannelCreateBiDir	Pointer to a tOCT6100_CHANNEL_CREATE_BIDIR structure. The structure's elements are defined below. The user allocates this structure.

5.2.4.1 tOCT6100_CHANNEL_CREATE_BIDIR Structure

pulBiDirChannelHndl handle

This parameter returns the handle for the newly created bi-directional channel. This handle is a unique value that identifies the bi-directional channel in all future function calls that affects this bi-directional channel. The user allocates the memory for this pointer.

Direction: IN/OUT Type: PUINT32

Default: NULL

ulFirstChannelHndl handle

This is the handle of one of the two channels used to create the bi-directional channel. This value was returned by a call to **Oct6100ChannelOpen**.

Direction: IN Type: UINT32

Default: cOCT6100_INVALID_HANDLE

ulSecondChannelHndl handle

This is the handle of the second channel used to create the bi-directional channel. This value was returned by a call to **Oct6100ChannelOpen**.

Direction: IN Type: UINT32

Default: cOCT6100_INVALID_HANDLE

5.2.5 Oct6100ChannelDestroyBiDir

This function destroys a bi-directional channel.

Calling this function does not close any resource. The two channels that were used to create the channel remain open but the API severs the link between them.

Usage

```
#include "oct6100_api.h"

UINT32 Oct6100ChannelDestroyBiDirDef (
            tPOCT6100_CHANNEL_DESTROY_BIDIR     f_pChannelDestroyBiDir
            );

UINT32 Oct6100ChannelDestroyBiDir (
            tPOCT6100_INSTANCE_API              f_pApiInstance,
            tPOCT6100_CHANNEL_DESTROY_BIDIR     f_pChannelDestroyBiDir
            );
```

Parameters

f_pApiInstance	Pointer to an instance structure of the chip.
f_pChannelDestroyBiDir	Pointer to a tOCT6100_CHANNEL_DESTROY_BIDIR structure. The structure's elements are defined below. The user allocates this structure.

5.2.5.1 tOCT6100_CHANNEL_DESTROY_BIDIR Structure

ulBiDirChannelHndl handle

This is the handle of the bi-directional channel for the API to destroy. This value was returned by a call to **Oct6100ChannelCreateBiDir**.

Direction: IN Type: UINT32

Default: cOCT6100_INVALID_HANDLE

5.2.6 Oct6100ChannelBroadcastTsstAdd

This function is used to broadcast an output to multiple timeslots. It assigns a broadcast H.100 timeslot to one of the output port (ROUT or SOUT) of the echo channel specified by **ulChannelHndl**. TSSTs assigned to a port with this function cannot be modified by a call to **Oct6100ChannelModify**.

To remove a broadcast TSST from a port, the user must call the function **Oct6100ChannelBroadcastTsstRemove**.

Usage

```
#include "oct6100_api.h"

UINT32 Oct6100ChannelBroadcastTsstAddDef (
            tPOCT6100_CHANNEL_BROADCAST_TSST_ADD

            f_pChannelBroadcastTsstAdd );

UINT32 Oct6100ChannelAssignTsst(
            tPOCT6100_INSTANCE_API          f_pApiInstance,
            tPOCT6100_CHANNEL_BROADCAST_TSST_ADD

            f_pChannelBroadcastTsstAdd );
```

Parameters

f_pApiInstance — Pointer to an instance structure of the chip.

f_pChannelBroadcastTsstAdd — Pointer to a tOCT6100_CHANNEL_BROADCAST_TSST_ADD structure. The structure's elements are defined below. The user allocates this structure.

5.2.6.1 tOCT6100_CHANNEL_BROADCAST_TSST_ADD Structure

ulChannelHndl handle

Channel handle. This handle is returned by a call to **Oct6100ChannelOpen**.

Direction: IN Type: UINT32
Default: cOCT6100_INVALID_HANDLE

ulPort cOCT6100_CHANNEL_PORT_ROUT
cOCT6100_CHANNEL_PORT_SOUT

This parameter represents the port on which the broadcast TSST will be attached.

Direction: IN Type: UINT32
Default: cOCT6100_INVALID_PORT

ulTimeslot

0 – 255 for 16 MHz stream frequency
0 – 127 for 8 MHz stream frequency
0 – 63 for 4 MHz stream frequency
0 – 31 for 2 MHz stream frequency

This is the H.100 timeslot of the channel's selected port. Note that allowed values are affected by the frequency of the clock that controls the **ulStream**.

Direction: IN Type: UINT32

Default: cOCT6100_INVALID_TIMESLOT

ulStream

0 – 31 for **ulMaxTdmStreams** of 32
0 – 15 for **ulMaxTdmStreams** of 16
0 – 7 for **ulMaxTdmStreams** of 8
0 – 3 for **ulMaxTdmStreams** of 4

This is the TDM stream of the channel's selected port. Note that allowed values are also affected by the **ulMaxTdmStreams** value specified at the **Oct6100ChipOpen** call.

Direction: IN Type: UINT32

Default: cOCT6100_INVALID_STREAM

5.2.7 Oct6100ChannelBroadcastTsstRemove

This function removes the bound created between an H.100 timeslot and one of the ports (ROUT or SOUT) of the echo channel specified by **ulChannelHndl**.

Usage

#include "oct6100_api.h"

UINT32 Oct6100ChannelBroadcastTsstRemoveDef (
tPOCT6100_CHANNEL_BROADCAST_TSST_REMOVE

f_pChanBroadcastTsstRemove);

UINT32 Oct6100ChannelReleaseTsst (
tPOCT6100_INSTANCE_API f_pApiInstance,
tPOCT6100_CHANNEL_BROADCAST_TSST_REMOVE

f_pChanBroadcastTsstRemove);

Parameters

f_pApiInstance Pointer to an instance structure of the chip.

f_pChanBroadcastTsstRemove Pointer to a tOCT6100_CHANNEL_BROADCAST_TSST_REMOVE structure. The structure's elements are defined below. The user allocates this structure.

5.2.7.1 tOCT6100_CHANNEL_BROADCAST_TSST_REMOVE Structure

ulChannelHndl handle

Channel's handle. This handle is returned by a call to **Oct6100ChannelOpen**.

Direction: IN Type: UINT32

Default: cOCT6100_INVALID_HANDLE

ulPort cOCT6100_CHANNEL_PORT_ROUT
cOCT6100_CHANNEL_PORT_SOUT

This parameter represents the port on which the broadcast TSST is attached.

Direction: IN Type: UINT32

Default: cOCT6100_INVALID_PORT

ulTimeslot 0 – 255 for 16 MHz stream frequency
0 – 127 for 8 MHz stream frequency
0 – 63 for 4 MHz stream frequency
0 – 31 for 2 MHz stream frequency

This is the H.100 timeslot of the channel's selected port. Note that allowed values are affected by the frequency of the clock that controls the **ulStream**.

This parameter is ignored if **fRemoveAll** is set to TRUE.

Direction: IN Type: UINT32

Default: cOCT6100_INVALID_TIMESLOT

ulStream 0 – 31 for **ulMaxTdmStreams** of 32
0 – 15 for **ulMaxTdmStreams** of 16
0 – 7 for **ulMaxTdmStreams** of 8
0 – 3 for **ulMaxTdmStreams** of 4

This is the TDM stream of the channel's selected port. Note that allowed values are also affected by the **ulMaxTdmStreams** value specified at the **Oct6100ChipOpen** call.

This parameter is ignored if **fRemoveAll** is set to TRUE.

Direction: IN Type: UINT32

Default: cOCT6100_INVALID_STREAM

fRemoveAll TRUE/FALSE

If TRUE, all broadcast TSSTs present on the selected output port of the channel will be removed.

Direction: IN Type: BOOL

Default: FALSE

5.2.8 Oct6100ChannelMute

This function mutes the selected ports of the echo channel specified by **ulChannelHndl**.

Usage

#include "oct6100_api.h"

UINT32 Oct6100ChannelMuteDef (
tPOCT6100_CHANNEL_MUTE f_pChannelMute);

UINT32 Oct6100ChannelMute (
tPOCT6100_INSTANCE_API f_pApiInstance,
tPOCT6100_CHANNEL_MUTE f_pChannelMute);

Parameters

f_pApiInstance	Pointer to an instance structure of the chip.
f_pChannelMute	Pointer to a tOCT6100_CHANNEL_MUTE structure. The structure's elements are defined below. The user allocates this structure.

5.2.8.1 tOCT6100_CHANNEL_MUTE Structure

ulChannelHndl handle

Channel's handle. This handle is returned by a call to **Oct6100ChannelOpen**.

Direction: IN Type: UINT32

Default: cOCT6100_INVALID_HANDLE

ulPortMask

cOCT6100_CHANNEL_MUTE_PORT_RIN
cOCT6100_CHANNEL_MUTE_PORT_ROUT
cOCT6100_CHANNEL_MUTE_PORT_SIN
cOCT6100_CHANNEL_MUTE_PORT_SIN_WITH_FEATURES
cOCT6100_CHANNEL_MUTE_PORT_SOUT
cOCT6100_CHANNEL_MUTE_NONE

This parameter represents the port mask on which muting should be applied. Many ports can be muted by ORing the required defines together.

If this function is called more than once, the value for **ulPortMask** is not replaced, but accumulated. For example, if RIN is masked on the first function call, and ROUT and SOUT are masked on the second function call, then RIN, ROUT and SOUT will now be masked. To remove a mask, the un-mute function must be used.

The cOCT6100_CHANNEL_MUTE_PORT_SIN_WITH_FEATURES port mask will mute the signal on the SIN port, but will allow features such as buffer playout and tone detection on that port to continue working.

Direction: IN Type: UINT32

Default: cOCT6100_CHANNEL_MUTE_NONE

5.2.9 Oct6100ChannelUnMute

This function un-mutes the selected ports of the echo channel specified by **ulChannelHndl**.

Usage

#include "oct6100_api.h"

```
UINT32 Oct6100ChannelUnMuteDef (
            tPOCT6100_CHANNEL_UNMUTE        f_pChannelUnMute );

UINT32 Oct6100ChannelUnMute (
            tPOCT6100_INSTANCE_API          f_pApiInstance,
            tPOCT6100_CHANNEL_UNMUTE        f_pChannelUnMute );
```

Parameters

f_pApiInstance	Pointer to an instance structure of the chip.
f_pChannelUnMute	Pointer to a tOCT6100_CHANNEL_UNMUTE structure. The structure's elements are defined below. The user allocates this structure.

5.2.9.1 tOCT6100_CHANNEL_UNMUTE Structure

ulChannelHndl handle

Channel's handle. This handle is returned by a call to **Oct6100ChannelOpen**.

Direction: IN Type: UINT32

Default: cOCT6100_INVALID_HANDLE

ulPortMask

cOCT6100_CHANNEL_MUTE_PORT_RIN
cOCT6100_CHANNEL_MUTE_PORT_ROUT
cOCT6100_CHANNEL_MUTE_PORT_SIN
cOCT6100_CHANNEL_MUTE_PORT_SIN_WITH_FEATURES
cOCT6100_CHANNEL_MUTE_PORT_SOUT
cOCT6100_CHANNEL_MUTE_PORT_NONE

This parameter represents the port mask on which un-muting should be applied. Many ports can be unmuted by ORing the required defines together.

Direction: IN Type: UINT32

Default: cOCT6100_CHANNEL_MUTE_NONE

5.2.10 Oct6100ChannelGetStats

This function retrieves the channel-related statistics.

Usage

```
#include "oct6100_api.h"

UINT32 Oct6100ChannelGetStatsDef (
        tPOCT6100_CHANNEL_STATS         f_pChannelStats );

UINT32 Oct6100ChannelGetStats (
        tPOCT6100_INSTANCE_API          f_pApiInstance,
        tPOCT6100_CHANNEL_STATS         f_pChannelStats );
```

Parameters

f_pApiInstance	Pointer to an instance structure of the chip.
f_pChannelStats	Pointer to a tOCT6100_CHANNEL_STATS structure. The structure's elements are defined below. The user allocates this structure.

5.2.10.1 tOCT6100_CHANNEL_STATS Structure

ulChannelHndl handle

The channel's handle for which the statistics are requested. This value was returned by a call to **Oct6100ChannelOpen**.

Direction: IN Type: UINT32
Default: cOCT6100_INVALID_HANDLE

fResetStats TRUE / FALSE

If TRUE, the API resets the following channel statistics: lMaxERL, lMaxERLE and ulMaxEchoDelay.

Direction: IN Type: BOOL
Default: FALSE

ulUserChanId See tOCT6100_CHANNEL_OPEN structure.

ulEchoOperationMode See tOCT6100_CHANNEL_OPEN structure.

fEnableToneDisabler See tOCT6100_CHANNEL_OPEN structure.

ulMutePortsMask

cOCT6100_CHANNEL_MUTE_PORT_RIN
cOCT6100_CHANNEL_MUTE_PORT_ROUT
cOCT6100_CHANNEL_MUTE_PORT_SIN
cOCT6100_CHANNEL_MUTE_PORT_SIN_WITH_FEATURES
cOCT6100_CHANNEL_MUTE_PORT_SOUT
cOCT6100_CHANNEL_MUTE_NONE

Ports that are currently muted, resulting from a call to **Oct6100ChannelMute**.

Direction: OUT Type: UINT32
Default: cOCT6100_CHANNEL_MUTE_NONE

fEnableExtToneDetection See tOCT6100_CHANNEL_OPEN structure.

lCurrentERL -127 – 127 dB

Current Echo Return Loss.

ERL defines the echo return loss estimated by the echo canceller. This value becomes valid as the Adaptive Filter is converged. In technical terms, it is the decrease in power of the receive signal as it passes through the local path hybrid and returns to the echo canceller on the send path through Sin. This parameter is measured over a certain period of time

The API returns cOCT6100_INVALID_SIGNED_STAT if the channel is not converged.

Direction: OUT Type: INT32

Default: cOCT6100_INVALID_SIGNED_STAT

lCurrentERLE -127 – 127 dB

Current Echo Return Loss Enhancement.

ERLE refers to the attenuation of the echo signal as it passes through the Send Path (Send In to Send Out) of the echo canceller. This specifically excludes any non-linear processing on the output of the canceller to provide further attenuation.

The API returns cOCT6100_INVALID_SIGNED_STAT if the channel is not converged.

Direction: OUT Type: INT32

Default: cOCT6100_INVALID_SIGNED_STAT

lMaxERL -127 – 127 dB

Maximum value of the ERL since the last reset or in the last measurement period.

The API returns cOCT6100_INVALID_SIGNED_STAT if the channel is not converged.

Direction: OUT Type: INT32

Default: cOCT6100_INVALID_SIGNED_STAT

lMaxERLE -127 – 127 dB

Maximum value of the ERLE since the last reset or in the last measurement period.

The API returns cOCT6100_INVALID_SIGNED_STAT if the channel is not converged.

Direction: OUT Type: INT32

Default: cOCT6100_INVALID_SIGNED_STAT

ulNumEchoPathChanges 32-bits value

This counter is incremented when a change is detected on the echo path.

Direction: OUT Type: UINT32

Default: cOCT6100_INVALID_STAT

ulCurrentEchoDelay 32-bits value

The delay in milliseconds at which the algorithm has detected energy on Sin correlated to the receive path. Technically, this defines the furthest in the H register that significant values are found.

The API returns cOCT6100_INVALID_STAT if the channel is not converged.

Direction: OUT Type: UINT32

Default: cOCT6100_INVALID_STAT

ulMaxEchoDelay 32-bits value

The longest delay in milliseconds at which the algorithm has found energy on Sin correlated to the receive path since the last reset of the statistics. Technically, this defines the furthest in the H register that significant values are found.

The API returns cOCT6100_INVALID_STAT if the channel is not converged.

Direction: OUT Type: UINT32

Default: cOCT6100_INVALID_STAT

ulToneDisablerStatus cOCT6100_TONE_DISABLER_EC_DISABLED
cOCT6100_TONE_DISABLER_EC_ENABLED

Status of the tone disabler for the current channel. This parameter will only return cOCT6100_TONE_DISABLER_EC_DISABLED if the tone disabler is enabled (**fEnableToneDisabler** has been set to TRUE) and a 2100 Hz tone is present for the requested channel.

If the tone disabler is enabled and the tone detector detects a 2100 Hz tone with phase reversals on the channel, both echo cancellation and the NLP will be disabled. If the tone disabler is enabled and a 2100 Hz tone without phase reversals is detected on the channel, only the NLP will be disabled on the channel. In both cases, the API will return cOCT6100_TONE_DISABLER_EC_DISABLED.

If there is no 2100 Hz tone with or without phase reversals, the tone disabler will not disable echo cancellation or the NLP on the channel. In this case, the API will return cOCT6100_TONE_DISABLER_EC_ENABLED.

Direction: OUT Type: UINT32

Default: cOCT6100_INVALID_STAT

fSinVoiceDetected TRUE / FALSE

If TRUE, voice activity is currently detected on the Sin port signal. This feature is available only with OCT61x6 devices. This will always be FALSE when running the API on OCT61x2 or OCT61x4 devices..

Direction: OUT Type: BOOL

Default: FALSE

fEchoCancellerConverged TRUE / FALSE

If TRUE, the echo canceller has detected and converged on an echo path and is removing echo.

The API returns FALSE if the channel is not converged.

Direction: OUT Type: BOOL

Default: FALSE

lRinLevel -127 – 127 dBm0

Average power of the signal level on the Rin port.

Direction: OUT Type: INT32

Default: cOCT6100_INVALID_SIGNED_STAT

lSinLevel -127 – 127 dBm0

Average power of the signal level on the Sin port.

Direction: OUT Type: INT32

Default: cOCT6100_INVALID_SIGNED_STAT

lRinAppliedGain -24 – 24 dB

Current gain applied to the signal level on the Rin port.

Direction: OUT Type: INT32

Default: cOCT6100_INVALID_SIGNED_STAT

lSoutAppliedGain -24 – 24 dB

Current gain applied to the signal level on the Sout port.

Direction: OUT Type: INT32

Default: cOCT6100_INVALID_SIGNED_STAT

lComfortNoiseLevel -127 – 127 dBm0

Average power of the comfort noise injected.

Direction: OUT Type: INT32

Default: cOCT6100_INVALID_SIGNED_STAT

TdmConfig structure

This structure contains all the configurations and statistics related to the TDM interface of a channel.

Direction: OUT Type: tOCT6100_CHANNEL_STATS_TDM

Default: see structure description

VqeConfig structure

This structure contains all the configurations and statistics related to the voice quality enhancement parameters.

Direction: OUT Type: tOCT6100_CHANNEL_STATS_VQE

Default: see structure description

CodecConfig structure

This structure contains all the configurations and statistics related to the encoder/decoder related parameters.

Direction: OUT Type: tOCT6100_CHANNEL_STATS_CODEC

Default: see structure description

5.2.10.2 tOCT6100_CHANNEL_STATS_TDM Structure

ulSinPcmLaw	See tOCT6100_CHANNEL_OPEN_TDM structure.
ulSinNumTssts	See tOCT6100_CHANNEL_OPEN_TDM structure.
ulSinTimeslot	See tOCT6100_CHANNEL_OPEN_TDM structure.
ulSinStream	See tOCT6100_CHANNEL_OPEN_TDM structure.
ulRinPcmLaw	See tOCT6100_CHANNEL_OPEN_TDM structure.
ulRinNumTssts	See tOCT6100_CHANNEL_OPEN_TDM structure.
ulRinTimeslot	See tOCT6100_CHANNEL_OPEN_TDM structure.
ulRinStream	See tOCT6100_CHANNEL_OPEN_TDM structure.
ulSoutPcmLaw	See tOCT6100_CHANNEL_OPEN_TDM structure.
ulSoutNumTssts	See tOCT6100_CHANNEL_OPEN_TDM structure.
ulSoutTimeslot	See tOCT6100_CHANNEL_OPEN_TDM structure.
ulSoutStream	See tOCT6100_CHANNEL_OPEN_TDM structure.
ulRoutPcmLaw	See tOCT6100_CHANNEL_OPEN_TDM structure.
ulRoutNumTssts	See tOCT6100_CHANNEL_OPEN_TDM structure.
ulRoutTimeslot	See tOCT6100_CHANNEL_OPEN_TDM structure.
ulRoutStream	See tOCT6100_CHANNEL_OPEN_TDM structure.
ulMaxBroadcastTssts	0 – 4096

This parameter defines the maximum number of entry allowed in the broadcast TSSTs arrays **pulRoutBroadcastStream**, **pulRoutBroadcastTimeslot**, **pulSoutBroadcastStream** and **pulSoutBroadcastTimeslot**.

The actual number of valid entry returned by the API will be specified by **ulNumRoutBroadcastTssts** and **ulNumSoutBroadcastTssts**.

Direction: OUT Type: UINT32

Default: 0

ulNumRoutBroadcastTssts 0 – **ulMaxBroadcastTssts**

This parameter defines the number of H.100 TDM timeslot associated to the ROUT port of the echo channel returned by this function call. Since broadcasting is supported on output ports, more then one TSST can be associated to the ROUT port.

The number of valid entries present in the arrays **pulRoutBroadcastTimeslot** and **pulRoutBroadcastStream** is defined by this parameter.

Direction: OUT Type: UINT32

Default: 0

ulNumSoutBroadcastTssts 0 – **ulMaxBroadcastTssts**

This parameter defines the number of H.100 TDM timeslot associated to the SOUT port of the echo channel returned by this function call. Since broadcasting is supported on output ports, more then one TSST can be associated to the SOUT port.

The number of valid entries present in the arrays **pulSoutBroadcastTimeslot** and **pulSoutBroadcastStream** is defined by this parameter.

Direction: OUT Type: UINT32

Default: 0

pulSoutBroadcastTimeslot array.

This contains valid H.100 TDM timeslot values associated to the SOUT port of the echo channel. The user allocates the memory for this array. Its size is specified by **ulMaxOutputTssts**.

Direction: OUT Type: PUINT32

Default: NULL

pulSoutBroadcastStream array.

This contains valid H.100 TDM steam values associated to the SOUT port of the echo channel. The user allocates the memory for this array. Its size is specified by **ulMaxBroadcastTssts**.

Direction: OUT Type: PUINT32

Default: NULL

pulRoutBroadcastTimeslot array

This contains valid H.100 TDM timeslot values associated to the ROUT port of the echo channel. The user allocates the memory for this array. Its size is specified by **ulMaxBroadcastTssts**.

Direction: OUT Type: PUINT32

Default: NULL

pulRoutBroadcastStream array.

This contains valid H.100 TDM steam values associated to the ROUT port of the echo channel. The user allocates the memory for this array. Its size is specified by **ulMaxBroadcastTssts**.

Direction: OUT Type: PUINT32

Default: NULL

fMoreRoutBroadcastTssts TRUE / FALSE

If TRUE, not all ROUT broadcast TSSTs were returned during this function call because **ulMaxBroadcastTssts** was too small.

Direction: OUT Type: BOOL

Default: FALSE

fMoreSoutBroadcastTssts TRUE / FALSE

If TRUE, not all SOUT broadcast TSSTs were returned during this function call because **ulMaxBroadcastTssts** was too small.

Direction: OUT Type: BOOL

Default: FALSE

5.2.10.3 tOCT6100_CHANNEL_STATS_VQE Structure

fEnableNlp See tOCT6100_CHANNEL_OPEN_VQE structure.

fEnableTailDisplacement See tOCT6100_CHANNEL_OPEN_VQE structure.

ulTailDisplacement See tOCT6100_CHANNEL_OPEN_VQE structure.

ulTailLength See tOCT6100_CHANNEL_OPEN_VQE structure.

fSinDcOffsetRemoval See tOCT6100_CHANNEL_OPEN_VQE structure.

fRinDcOffsetRemoval See tOCT6100_CHANNEL_OPEN_VQE structure.

fRinLevelControl See tOCT6100_CHANNEL_OPEN_VQE structure.

lRinLevelControlGainDb See tOCT6100_CHANNEL_OPEN_VQE structure.

fSoutLevelControl See tOCT6100_CHANNEL_OPEN_VQE structure.

lSoutLevelControlGainDb See tOCT6100_CHANNEL_OPEN_VQE structure.

fRinAutomaticLevelControl See tOCT6100_CHANNEL_OPEN_VQE structure.

lRinAutomaticLevelControlTargetDb See tOCT6100_CHANNEL_OPEN_VQE structure.

fSoutAutomaticLevelControl See tOCT6100_CHANNEL_OPEN_VQE structure.

lSoutAutomaticLevelControlTargetDb See tOCT6100_CHANNEL_OPEN_VQE structure.

fRinHighLevelCompensation See tOCT6100_CHANNEL_OPEN_VQE structure.

lRinHighLevelCompensationThresholdDb See tOCT6100_CHANNEL_OPEN_VQE structure.

fSoutAdaptiveNoiseReduction See tOCT6100_CHANNEL_OPEN_VQE structure.

fSoutNoiseBleaching	See tOCT6100_CHANNEL_OPEN_VQE structure.
fSoutConferencingNoiseReduction	See tOCT6100_CHANNEL_OPEN_VQE structure.
ulComfortNoiseMode	See tOCT6100_CHANNEL_OPEN_VQE structure.
fDtmfToneRemoval	See tOCT6100_CHANNEL_OPEN_VQE structure.
fAcousticEcho	See tOCT6100_CHANNEL_OPEN_VQE structure.
ulNonLinearityBehaviorA	See tOCT6100_CHANNEL_OPEN_VQE structure.
ulNonLinearityBehaviorB	See tOCT6100_CHANNEL_OPEN_VQE structure.
ulDoubleTalkBehavior	See tOCT6100_CHANNEL_OPEN_VQE structure.
lDefaultErlDb	See tOCT6100_CHANNEL_OPEN_VQE structure.
lAecDefaultErlDb	See tOCT6100_CHANNEL_OPEN_VQE structure.
ulAecTailLength	See tOCT6100_CHANNEL_OPEN_VQE structure.
ulSoutAutomaticListenerEnhancementGainDb	See tOCT6100_CHANNEL_OPEN_VQE structure.
fSoutNaturalListenerEnhancement	See tOCT6100_CHANNEL_OPEN_VQE structure.
ulSoutNaturalListenerEnhancementGainDb	See tOCT6100_CHANNEL_OPEN_VQE structure.
lAnrSnrEnhancementDb	See tOCT6100_CHANNEL_OPEN_VQE structure.
ulAnrVoiceNoiseSegregation	See tOCT6100_CHANNEL_OPEN_VQE structure.
ulToneDisablerVqeActivationDelay	See tOCT6100_CHANNEL_OPEN_VQE structure.
fEnableMusicProtection	See tOCT6100_CHANNEL_OPEN_VQE structure.

fIdleCodeDetection	See tOCT6100_CHANNEL_OPEN_VQE structure.

5.2.10.4 tOCT6100_CHANNEL_STATS_CODEC Structure

ulAdpcmNibblePosition	See tOCT6100_CHANNEL_OPEN_CODEC structure.
ulEncoderPort	See tOCT6100_CHANNEL_OPEN_CODEC structure.
ulEncodingRate	See tOCT6100_CHANNEL_OPEN_CODEC structure.
ulDecoderPort	See tOCT6100_CHANNEL_OPEN_CODEC structure.
ulDecodingRate	See tOCT6100_CHANNEL_OPEN_CODEC structure.
fEnableSilenceSuppression	See tOCT6100_CHANNEL_OPEN_CODEC structure.
ulPhasingType	See tOCT6100_CHANNEL_OPEN_CODEC structure.
ulPhase	See tOCT6100_CHANNEL_OPEN_CODEC structure.
ulPhasingTsstHndl	See tOCT6100_CHANNEL_OPEN_CODEC structure.

5.3 Conference Bridge Functions

Conference bridge functions are used to open, close, and monitor the conference bridge structure. Functions to add and remove channels to a conference bridge are also described in this section.

A **simple** conference bridge mixes all the SOUT or RIN port signals of the channels present on the bridge. Each channel will receive the resulting signal (minus its own signal) as its RIN port signal.

In the case of 512- and 672-channel devices, some limitations apply. Rin conferencing can be applied on up to 671 channels. The number of channels of Sout conferencing is limited to 447, however using this many mixers reduces the total device capacity to 542 channels. If 672 channels of echo cancellation are required, then only 188 channels can also support Sout conferencing.

The following table indicates the number of Sout conferencing channels that can be supported by the 512- and 672-channel devices.

Device Capacity	Number of channels performing only echo cancellation	Number of channels performing echo cancellation and Sout conferencing	Total channels
512	65	447 (max)	512
672	484	188	672 (max)
672	95	447 (max)	542

Limitations on conference bridges are based on the number of available mixer and TSI resources. The total available mixer resources is 1342 and the total available TSI resources is 1532. The preceding table's values are calculated like this:

For Rin conferencing participants:

- Mixer resources required = total nb of participants * 2
- TSI resources required = total nb of participants * 2

For Sout conferencing participants:

- Mixer resources required = total nb of participants * 3
- TSI resources required = total nb of participants * 3

A **flexible** conference bridge can be used to mask the signal of certain participants from other participants, i.e. each channel will receive the sum of all the channels present on the bridge minus its own and the masked channels. These flexible bridges are limited to 32 participants. This is typically used for call monitoring, or "coaching" where a third party wishes to hear many participants, but only be heard by one participant. The equations to calculate resource allocation for flexible conference bridges are the following:

For Rin flexible conferencing participants:

- Mixer resources required =

  nb of Conference Bridges * (nb of part. per bridge * (nb of part. per bridge + 1))
- TSI resources required = total nb of participants * 3

For Sout flexible conferencing participants:

- Mixer resources required =

nb of Conference Bridges * (nb of part. per bridge * (nb of part. per bridge + 1) - nb of part. per bridge)

- TSI resources required = total nb of participants * 3

A channel performing ADPCM compression or decompression **cannot** be used for conferencing.

5.3.1 Oct6100ConfBridgeOpen

This function opens a conference bridge. Initially, there is no channel connected to the bridge. Channels are added to the bridge by calling **Oct6100ConfBridgeAddChan**. A channel is removed from the bridge by calling the function **Oct6100ConfBridgeRemoveChan**.

This function returns a handle by which the API identifies this bridge.

Usage

```
#include "oct6100_api.h"

UINT32 Oct6100ConfBridgeOpenDef (
            tPOCT6100_CONF_BRIDGE_OPEN      f_pConfBridgeOpen );

UINT32 Oct6100ConfBridgeOpen (
            tPOCT6100_INSTANCE_API          f_pApiInstance,
            tPOCT6100_CONF_BRIDGE_OPEN      f_pConfBridgeOpen );
```

Parameters

f_pApiInstance	Pointer to an instance structure of the chip.
f_pConfBridgeOpen	Pointer to a tOCT6100_CONF_BRIDGE_OPEN structure. The structure's elements are defined below. The user allocates this structure.

5.3.1.1 tOCT6100_CONF_BRIDGE_OPEN Structure

pulConfBridgeHndl handle

The parameter returns the handle for the newly created conference bridge. This handle is a unique value that identifies the bridge in all future function calls affecting this bridge. The user allocates the memory for this pointer.

Direction: IN/OUT Type: PUINT32

Default: NULL

fFlexibleConferencing TRUE / FALSE

If TRUE, flexible conferencing is enabled. In this mode, the user can choose which signals are masked for every participant in the conference. Enabling this mode will limit the number of participants to be added to the conference bridge to 32.

Direction: IN Type: BOOL

Default: FALSE

5.3.2 Oct6100ConfBridgeClose

This function closes a conference bridge.

A conference bridge can only be closed if no channels are present on the bridge.

Usage

#include "oct6100_api.h"

```
UINT32 Oct6100ConfBridgeCloseDef (
        tPOCT6100_CONF_BRIDGE_CLOSE     f_pConfBridgeClose );

UINT32 Oct6100ConfBridgeClose (
        tPOCT6100_INSTANCE_API          f_pApiInstance,
        tPOCT6100_CONF_BRIDGE_CLOSE     f_pConfBridgeClose );
```

Parameters

f_pApiInstance	Pointer to an instance structure of the chip.
f_pConfBridgeClose	Pointer to a tOCT6100_CONF_BRIDGE_CLOSE structure. The structure's elements are defined below. The user allocates this structure.

5.3.2.1 tOCT6100_CONF_BRIDGE_CLOSE Structure

ulConfBridgeHndl handle

Handle of the conference bridge to be closed. This value is returned by a call to **Oct6100ConfBridgeOpen**.

Direction: IN Type: UINT32

Default: cOCT6100_INVALID_HANDLE

5.3.3 Oct6100ConfBridgeChanAdd

This function adds a channel to an already opened conference bridge. The conference bridge and channel handles must be valid to perform a valid addition.

A channel can only be part of one conference bridge at a time. To move a channel from a bridge to another, the user must remove the channel from the first bridge and add it to the second.

Usage

```
#include "oct6100_api.h"

UINT32 Oct6100ConfBridgeChanAddDef (
            tPOCT6100_CONF_BRIDGE_CHAN_ADD      f_pConfBridgeAdd );

UINT32 Oct6100ConfBridgeChanAdd (
            tPOCT6100_INSTANCE_API              f_pApiInstance,
            tPOCT6100_CONF_BRIDGE_CHAN_ADD      f_pConfBridgeAdd );
```

Parameters

f_pApiInstance	Pointer to an instance structure of the chip.
f_pConfBridgeAdd	Pointer to a tOCT6100_CONF_BRIDGE_ADD_CHAN structure. The structure's elements are defined below. The user allocates this structure.

5.3.3.1 tOCT6100_CONF_BRIDGE_CHAN_ADD Structure

ulConfBridgeHndl handle

Handle of the conference bridge. This value is returned by a call to **Oct6100ConfBridgeOpen**.

Direction: IN Type: UINT32

Default: cOCT6100_INVALID_HANDLE

ulChannelHndl handle

Handle of the echo cancellation channel to be added to the conference bridge. This value is returned by a call to **Oct6100ChannelOpen**.

Direction: IN Type: UINT32

Default: cOCT6100_INVALID_HANDLE

ulInputPort cOCT6100_CHANNEL_PORT_SOUT
cOCT6100_CHANNEL_PORT_RIN

This parameter indicates which channel port of the channel will be added to the conference bridge.

Direction: IN Type: UINT32

Default: cOCT6100_CHANNEL_PORT_SOUT

ulListenerMaskIndex 0 - 31

When flexible conference bridges are enabled, this is the index of the current channel. This Index or channel number is used to identify the listener in the **ulListenerMask** parameter.

Direction: IN Type: UINT32

Default: cOCT6100_INVALID_VALUE

ulListenerMask 32-bit value

When flexible conference bridges are enabled, this bit mask dictates which conference bridge participants cannot be heard by this channel. The user sets the bit at position **ulListenerMaskIndex** to subtract the signal from the specified participant. For example, setting the mask to 0x8 would remove channel with **ulListenerMaskIndex** equal to 3 from the mixed signal for this channel.

Direction: IN Type: UINT32

Default: 0x0

fMute TRUE / FALSE

If TRUE, this channel will be added muted to the conference bridge.

Direction: IN Type: BOOL

Default: FALSE

ulTappedChannelHndl handle

This parameter is used when the channel being added to the conference bridge is to perform lawful interception. When this parameter is not set to cOCT6100_INVALID_HANDLE, it represents the echo cancellation channel to be tapped in the conference bridge.

When this parameter is used, the participant added to the bridge will hear the SOUT ports of all participants, as well as the ROUT port of the tapped participant.

The **ulRinStream** and **ulRinTimeslot** parameters of the TDM configuration of the channel specified by this handle must be set to **cOCT6100_UNASSIGNED** for tapping to work correctly. This is done automatically when adding the participant, but the user should take care of never modifying the channel to re-assign the TSST.

An error will be returned by the API if the user tries to remove a tapped channel from a conference bridge. The channel tapping the participant should be removed first.

Direction: IN Type: UINT32

Default: cOCT6100_INVALID_HANDLE

5.3.4 Oct6100ConfBridgeChanRemove

This function removes a channel from a conference bridge.

Usage

```
#include "oct6100_api.h"

UINT32 Oct6100ConfBridgeChanRemoveDef (
            tPOCT6100_CONF_BRIDGE_CHAN_REMOVE    f_pConfBridgeRemove );

UINT32 Oct6100ConfBridgeChanRemove (
            tPOCT6100_INSTANCE_API               f_pApiInstance,
            tPOCT6100_CONF_BRIDGE_CHAN_REMOVE    f_pConfBridgeRemove );
```

Parameters

f_pApiInstance	Pointer to an instance structure of the chip.
f_pConfBridgeRemove	Pointer to a tOCT6100_CONF_BRIDGE_CHAN_REMOVE structure. The structure's elements are defined below. The user allocates this structure.

5.3.4.1 tOCT6100_CONF_BRIDGE_CHAN_REMOVE Structure

ulConfBridgeHndl handle

Handle of the conference bridge from which the channel is removed. This value is returned by a call to **Oct6100ConfBridgeOpen**. This parameter is ignored if **fRemoveAll** is set to FALSE.

Direction: IN Type: UINT32

Default: cOCT6100_INVALID_HANDLE

ulChannelHndl handle

Handle of the echo cancellation channel to be removed from the conference bridge. This value is returned by a call to **Oct6100ChannelOpen**. This parameter is ignored if **fRemoveAll** is set to TRUE.

Direction: IN Type: UINT32

Default: cOCT6100_INVALID_HANDLE

fRemoveAll TRUE / FALSE

If TRUE, all channels currently on the conference bridge will be removed.

Direction: IN Type: BOOL

Default: FALSE

5.3.5 Oct6100ConfBridgeChanMute

This function will mute a channel currently on a conference bridge.

Usage

```
#include "oct6100_api.h"

UINT32 Oct6100ConfBridgeChanMuteDef (
            tPOCT6100_CONF_BRIDGE_CHAN_MUTE     f_pConfBridgeMute );

UINT32 Oct6100ConfBridgeChanMute (
            tPOCT6100_INSTANCE_API              f_pApiInstance,
            tPOCT6100_CONF_BRIDGE_CHAN_MUTE     f_pConfBridgeMute );
```

Parameters

f_pApiInstance	Pointer to an instance structure of the chip.
f_pConfBridgeMute	Pointer to a tOCT6100_CONF_BRIDGE_CHAN_MUTE structure. The structure's elements are defined below. The user allocates this structure.

5.3.5.1 tOCT6100_CONF_BRIDGE_CHAN_MUTE Structure

ulChannelHndl handle

Handle of the echo cancellation channel to be muted. This value is returned by a call to **Oct6100ChannelOpen**.

Direction: IN Type: UINT32

Default: cOCT6100_INVALID_HANDLE

5.3.6 Oct6100ConfBridgeChanUnMute

This function will un-mute a channel currently on a conference bridge. The channel will now be broadcast onto its conference bridge

Usage

#include "oct6100_api.h"

```
UINT32 Oct6100ConfBridgeChanUnMuteDef (
            tPOCT6100_CONF_BRIDGE_CHAN_UNMUTE    f_pConfBridgeUnMute );

UINT32 Oct6100ConfBridgeChanUnMute (
            tPOCT6100_INSTANCE_API               f_pApiInstance,
            tPOCT6100_CONF_BRIDGE_CHAN_UNMUTE    f_pConfBridgeUnMute );
```

Parameters

f_pApiInstance	Pointer to an instance structure of the chip.
f_pConfBridgeUnMute	Pointer to a tOCT6100_CONF_BRIDGE_CHAN_UNMUTE structure. The structure's elements are defined below. The user allocates this structure.

5.3.6.1 tOCT6100_CONF_BRIDGE_CHAN_UNMUTE Structure

ulChannelHndl handle

Handle of the echo cancellation channel to be un-muted. This value is returned by a call to **Oct6100ChannelOpen**.

Direction: IN Type: UINT32

Default: cOCT6100_INVALID_HANDLE

5.3.7 Oct6100ConfBridgeDominantSpeakerSet

This function determines which member of a conference bridge is the dominant speaker. The signal level of the other participants on the bridge is reduced when the dominant speaker is talking. This allows the dominant speaker to interrupt and talk-over whenever needed, while allowing feedback from other callers. This feature is available only with OCT61x6 devices. If this is executed on an OCT61x2 or OCT61x4 device, the function will return the error cOCT6100_ERR_NOT_SUPPORTED_DOMINANT_SPEAKER. Dominant speaker can only be set on a conferencing channel that uses Sout for its input port.

NOTE: The **fEnableNlp** and **fSoutConferencingNoiseReduction** variables of the channel must be set to TRUE for this feature to work properly.

Usage

```
#include "oct6100_api.h"

UINT32 Oct6100ConfBridgeDominantSpeakerSetDef (
            tPOCT6100_CONF_BRIDGE_DOMINANT_SPEAKER_SET
                                        f_pConfBridgeDominantSpeaker );

UINT32 Oct6100ConfBridgeDominantSpeakerSet (
            tPOCT6100_INSTANCE_API      f_pApiInstance,
            tPOCT6100_CONF_BRIDGE_DOMINANT_SPEAKER_SET
                                        f_pConfBridgeDominantSpeaker );
```

Parameters

f_pApiInstance Pointer to an instance structure of the chip.

f_pConfBridgeDominantSpeaker

Pointer to a tOCT6100_CONF_BRIDGE_DOMINANT_SPEAKER_SET structure. The structure's elements are defined below. The user allocates this structure.

5.3.7.1 tOCT6100_CONF_BRIDGE_DOMINANT_SPEAKER_SET Structure

ulConfBridgeHndl handle

This parameter is used to remove a dominant speaker attribute from a conference bridge.

This parameter is the handle of the conference bridge for which the dominant speaker is set. This value is returned by a call to **Oct6100ConfBridgeOpen**. This parameter is ignored if **ulChannelHndl** is given a value.

Direction: IN Type: UINT32

Default: cOCT6100_INVALID_HANDLE

ulChannelHndl handle

This parameter is used to add the dominant speaker attribute to a conference bridge participant.

This parameter is the handle of the echo cancellation channel that will be set as the dominant speaker of the conference bridge. This value is returned by a call to **Oct6100ChannelOpen**. To remove the dominant speaker attribute of a conference bridge participant, the **ulConfBridgeHndl** parameter is used, and this parameter should be set to **cOCT6100_CONF_NO_DOMINANT_SPEAKER_HNDL**.

Direction: IN Type: UINT32

Default: cOCT6100_INVALID_HANDLE

5.3.8 Oct6100ConfBridgeMaskChange

This function changes the listener mask of a flexible bridge participant.

Usage

```
#include "oct6100_api.h"

UINT32 Oct6100ConfBridgeMaskChangeDef (
            tPOCT6100_CONF_BRIDGE_MASK_CHANGE
                                                f_pConfBridgeMaskChange );

UINT32 Oct6100ConfBridgeMaskChange (
            tPOCT6100_INSTANCE_API          f_pApiInstance,
            tPOCT6100_CONF_BRIDGE_MASK_CHANGE
                                                f_pConfBridgeMaskChange );
```

Parameters

f_pApiInstance Pointer to an instance structure of the chip.

f_pConfBridgeMaskChange

Pointer to a tOCT6100_CONF_BRIDGE_MASK_CHANGE structure. The structure's elements are defined below. The user allocates this structure.

5.3.8.1 tOCT6100_CONF_BRIDGE_MASK_CHANGE Structure

ulChannelHndl handle

This parameter is the handle of the echo cancellation channel representing the flexible bridge participant where the listener mask will be updated.

Direction: IN Type: UINT32
Default: cOCT6100_INVALID_HANDLE

ulNewListenerMask 32-bit value

This parameter is the new listener mask to apply for the selected participant in the flexible bridge. Refer to the **ulListenerMask** member of the conference channel add function for a description of the listener mask.

Direction: IN Type: UINT32
Default: 0x0

5.3.9 Oct6100ConfBridgeGetStats

This function fills a tOCT6100_CONF_BRIDGE_STATS structure with the current statistics for the specified conference bridge. All statistics returned by this function are initialized by the **Oct6100ConfBridgeOpen** function.

Usage

```
#include "oct6100_api.h"

UINT32 Oct6100ConfBridgeGetStatsDef (
            tPOCT6100_CONF_BRIDGE_STATS     f_pConfBridgeStats );

UINT32 Oct6100ConfBridgeGetStats (
            tPOCT6100_INSTANCE_API          f_pApiInstance,
            tPOCT6100_CONF_BRIDGE_STATS     f_pConfBridgeStats );
```

Parameters

f_pApiInstance	Pointer to an instance structure of the chip
f_pConfBridgeStats	Pointer to a tOCT6100_CONF_BRIDGE_STATS statistics structure to be filled in by this routine. The structure's elements are defined below. The user allocates this structure.

5.3.9.1 tOCT6100_CONF_BRIDGE_STATS Structure

ulConfBridgeHndl handle

Handle of the conference bridge for which statistics are requested. This value is returned by a call to **Oct6100ConfBridgeOpen**.

Direction: IN Type: UINT32
Default: cOCT6100_INVALID_HANDLE

ulNumChannels 0 – 447

This is the number of channels currently present on the conference bridge. This count does not include the channels which are taps in the conference bridge.

Direction: OUT Type: UINT32
Default: cOCT6100_INVALID_STAT

ulNumTappedChannels 0 – 447

This is the number of channels currently being tapped on the conference bridge.

Direction: OUT Type: UINT32
Default: cOCT6100_INVALID_STAT

fFlexibleConferencing TRUE / FALSE

If TRUE, this is a flexible conference bridge.

Direction: OUT Type: BOOL
Default: cOCT6100_INVALID_STAT

5.4 Phasing TSST Functions

Phasing TSST functions are used for silence suppression and compression changes on an echo cancellation channel.

5.4.1 Oct6100PhasingTsstOpen

This function opens a phasing counter on a TDM H.1x0 timeslot. The TSST is used to synchronize compression rate changes by the OCT6100 device with packetization boundaries of a SAR device. Incremental values are driven by an external source (possibly the SAR device). The value range is between 0 and **ulPhasingLength** – 1. The configured **ulPhase** value (see tOCT6100_CHANNEL_OPEN structure) indicates that the external agent SAR is now fetching the first sample used to assemble the next packet. For example, a packet that consists of 40 samples of voice could be phased with a phasing TSST counting from 0 to 39.

With the phasing TSST and a phase, the OCT6100 device can determine the packetization boundary.

Phasing TSSTs are also used to allow silence suppression indications to be sent to the packetization device at the right moment. On the last byte of every packet (i.e. byte preceding packetization boundary) the chip can be configured to indicate whether the current packet should be suppressed. The device needs a phasing TSST from the SAR device to indicate the packetization boundaries.

The device supports up to 16 independent phasing TSSTs.

In cases where continuous (every H.1x0 frame) silence suppression and compression changes information is desired, the external agent should always drive a value of (**ulPhase** – 1) on the TDM timeslot used as the phasing counter. For example, if **ulPhasingLength** is 40 and **ulPhase** is 39, the external agent should continuously drive 38 on the TDM timeslot to receive silence suppression or compression changes information at every frame.

Usage

```
#include "oct6100_api.h"

UINT32 Oct6100PhasingTsstOpenDef (
            tPOCT6100_PHASING_TSST_OPEN     f_pPhasingTsstOpen );

UINT32 Oct6100PhasingTsstOpen (
            tPOCT6100_INSTANCE_API          f_pApiInstance,
            tPOCT6100_PHASING_TSST_OPEN     f_pPhasingTsstOpen );
```

Parameters

f_pApiInstance	Pointer to an instance structure of the chip
f_pPhasingTsstOpen	Pointer to a tOCT6100_PHASING_TSST_OPEN structure. The structure's elements are defined below.

5.4.1.1 tOCT6100_PHASING_TSST_OPEN Structure

pulPhasingTsstHndl handle

Pointer to a single UINT32 that returns the handle for the created phasing TSST. This handle is a unique value that identifies the phasing TSST in all future function calls affecting this phasing TSST. The user allocates the UINT32 for the handle.

Direction: IN/OUT Type: PUINT32

Default: NULL

ulPhasingLength 2 – 240

The external agent driving the phasing TSST will drive incremental values within the 0 - (**ulPhasingLength** – 1) range.

Direction: IN Type: UINT32

Default: 88

ulTimeslot 0 – 255 for 16 MHz stream frequency
0 – 127 for 8 MHz stream frequency
0 – 63 for 4 MHz stream frequency
0 – 31 for 2 MHz stream frequency

This is the timeslot component of the phasing H.100 timeslot. Note that allowed values are affected by the frequency of the clock that controls the **ulInputStream**.

Direction: IN Type: UINT32

Default: cOCT6100_INVALID_TIMESLOT

ulStream 0 – 31 for **ulMaxStream** of 32
0 – 15 for **ulMaxStream** of 16
0 – 7 for **ulMaxStream** of 8
0 – 3 for **ulMaxStream** of 4

This is the stream component of the phasing H.100 timeslot. Note that this value is also affected by the **ulMaxStream** value specified at the **Oct6100ChipOpen** call.

Direction: IN Type: UINT32

Default: cOCT6100_INVALID_STREAM

5.4.2 Oct6100PhasingTsstClose

This function closes the specified phasing TSST.

Usage

#include "oct6100_api.h"

UINT32 Oct6100PhasingTsstCloseDef (
tPOCT6100_PHASING_TSST_CLOSE f_pPhasingTsstClose);

UINT32 Oct6100PhasingTsstClose (
tPOCT6100_INSTANCE_API f_pApiInstance,
tPOCT6100_PHASING_TSST_CLOSE f_pPhasingTsstClose);

Parameters

f_pApiInstance	Pointer to an instance structure of the chip
f_pPhasingTsstClose	Pointer to a tOCT6100_PHASING_TSST_CLOSE structure. The structure's elements are defined below.

5.4.2.1 tOCT6100_PHASING_TSST_CLOSE Structure

ulPhasingTsstHndl handle

Handle that identifies the phasing TSST to be closed. This handle is returned by the **Oct6100PhasingTsstOpen** call that opened the phasing TSST.

Direction: IN Type: UINT32

Default: cOCT6100_INVALID_HANDLE

5.5 Tone Detection Functions

The OCT6100 firmware includes a tone profile, which defines the group of tones that can be detected on any open channel. Each tone of the profile is identified by its Tone Number, which determines the specific tone and detection port. The list of available tones is always provided with the firmware in an accompanying text file. For example, the text file contains lines such as the following:

#define ROUT_DTMF_1 0x10000011

This means that tone number 0x10000011 corresponds with the detection of the DTMF 1 tone on the Rout port. To enable the detection of that tone on a specific channel, the user must call the **Oct6100ToneDetectionEnable** function, passing it the targeted channel handle (returned from the call to **Oct6100ChannelOpen**) as well as 0x10000011, the Tone Number.

Once enabled, the channel will generate a tone event every time a DTMF 1 tone is detected on the Rout port. The user must call the **Oct6100InterruptServiceRoutine** regularly and check the tOCT6100_INTERRUPT_FLAGS parameter fToneEventsPending to see if any tone events are pending.

If fToneEventsPending is TRUE the user must call the **Oct6100EventGetTone** function to retrieve the pending tone events. A given channel may be configured to detect any combination of available tones by calling **Oct6100ToneDetectionEnable** multiple times with the ulToneNumber parameter set to the desired tone number.

The following table indicates delays before the generation of present and stop events. Those delays may vary depending on the energy and the frequency of the tone. A new tone is detected if there is a signal change longer than 10ms.

Tones	Delay before PRESENT event (ms)	Delay between PRESENT events (ms)	Delay before STOP event (ms)
2100HB_END	252	252	N/A
2100GB_ON	300 +/- 100	N/A	N/A
2100GB_WSPR	250	425*	N/A
1100GB_ON	425	N/A	20
SS5	30	N/A	20
SS7	400	N/A	20
DTMF	40	N/A	20
MF-R1	30	N/A	20
MF-R2	30	N/A	20

* The 2100Hz tone must continue for 50ms after the phase reversal to get the 2100GB_WSPR event.

WSPR = well space phase reversal

HB = Hold Band = silence

GB = Guard Band

5.5.1 Oct6100ToneDetectionEnable

This function allows the user to enable the detection of a preprogrammed tone on a channel.

Usage

#include "oct6100_api.h"

```
UINT32 Oct6100ToneDetectionEnableDef (
        tPOCT6100_TONE_DETECTION_ENABLE     f_pToneDetectEnable );

UINT32 Oct6100ToneDetectionEnable (
        tPOCT6100_INSTANCE_API              f_pApiInstance,
        tPOCT6100_TONE_DETECTION_ENABLE     f_pToneDetectEnable );
```

Parameters

f_pApiInstance	Pointer to an instance structure of the chip.
f_pToneDetectEnable	Pointer to a tOCT6100_TONE_DETECTION_ENABLE structure. The structure's elements are defined below. The user allocates this structure.

5.5.1.1 tOCT6100_TONE_DETECTION_ENABLE Structure

ulChannelHndl handle

The handle that identifies the channel for which tone detection is enabled. The handle is returned by the call to **Oct6100ChannelOpen**.

Direction: IN Type: UINT32

Default: cOCT6100_INVALID_HANDLE

ulToneNumber see **Tone Detection Functions Section**

Selects the tone to be detected on the specified channel.

Direction: IN Type: UINT32

Default: cOCT6100_INVALID_TONE

5.5.2 Oct6100ToneDetectionDisable

This function allows the user to disable the detection of a specific tone on a channel.

Usage

```
#include "oct6100_api.h"

UINT32 Oct6100ToneDetectionDisableDef (
            tPOCT6100_TONE_DETECTION_DISABLE        f_pToneDetectDisable );

UINT32 Oct6100ToneDetectionDisable (
            tPOCT6100_INSTANCE_API                  f_pApiInstance,
            tPOCT6100_TONE_DETECTION_DISABLE        f_pToneDetectDisable );
```

Parameters

f_pApiInstance	Pointer to an instance structure of the chip.
f_pToneDetectDisable	Pointer to a tOCT6100_TONE_DETECTION_DISABLE structure. The structure's elements are defined below. The user allocates this structure.

5.5.2.1 tOCT6100_TONE_DETECTION_DISABLE Structure

ulChannelHndl handle

The handle that identifies the channel for which tone detection is to be disabled. The handle is returned by the call to **Oct6100ChannelOpen**.

Direction: IN Type: UINT32

Default: cOCT6100_INVALID_HANDLE

ulToneNumber see **Tone Detection Functions Section**

Selects the tone to be disabled for the specified channel.

Direction: IN Type: UINT32

Default: cOCT6100_INVALID_TONE

fDisableAll TRUE / FALSE

Setting this flag to TRUE will disable tone detection on all enabled tones. The **ulToneNumber** parameter is ignored when this is set to TRUE.

Direction: IN Type: BOOL

Default: FALSE

5.6 Buffer Playout Functions

These functions are used to manage buffer playout on a channel. Three functions are used to manage the playout: add, start and stop functions. The **Oct6100BufferPlayoutAdd** function is used to add a buffer to the buffer list for the selected channel port. Up to 127 buffers can be added to a channel port.

Once all the buffers are added to the channel's list, they can be played out by calling **Oct6100BufferPlayoutStart**. The playout will stop once all buffers have been played out or when the function **Oct6100BufferPlayoutStop** is called by the user.

Other supporting functions are used to load or unload buffers into the chip's external memory: load, load in blocks and unload functions. This can be done at initialization or while the chip is operating.

Since loading buffers into the external memory of the chip can be a long process, depending on the size of the buffer to be loaded, the user is supplied with two types of load functions. **Oct6100BufferPlayoutLoad** loads the buffer directly into memory in one call, blocking the user application until this is done. **Oct6100BufferPlayoutLoadBlock** loads the buffer in blocks specified by the user, yielding control to the application after each block is loaded into external memory.

Note that the buffer playout function cannot be used on a channel that is also performing DTMF tone removal.

5.6.1 Oct6100BufferPlayoutLoad

This function allows the user to load a buffer into external memory.

Usage

```
#include "oct6100_api.h"

UINT32 Oct6100BufferPlayoutLoadDef (
            tPOCT6100_BUFFER_LOAD           f_pBufferLoad );

UINT32 Oct6100BufferPlayoutLoad (
            tPOCT6100_INSTANCE_API          f_pApiInstance,
            tPOCT6100_BUFFER_LOAD           f_pBufferLoad );
```

Parameters

f_pApiInstance	Pointer to an instance structure of the chip.
f_pBufferLoad	Pointer to a tOCT6100_BUFFER_LOAD structure. The structure's elements are defined below. The user allocates this structure.

5.6.1.1 tOCT6100_BUFFER_LOAD Structure

pulBufferIndex 0-1343

Buffer index in the buffer memory. This value is used in future **Oct6100BufferPlayout** function calls to reference this buffer.

Direction: OUT Type: PUINT32

Default: NULL

pulPlayoutFreeMemSize 0 - total space in external memory for playout

Optional parameter that returns the amount of external memory, in bytes, left that can be used for buffer playout after loading the specified buffer. Note that this value does not necessarily refer to a contiguous memory block.

Direction: OUT Type: PUINT32

Default: NULL

pbyBufferPattern pointer

A byte pointer pointing to a valid buffer to be loaded into the chip's external memory.

Direction: IN Type: PUINT8

Default: NULL

ulBufferSize 64 – max space left in external memory

Size of the buffer loaded into external memory. This value must be modulo 16. This size is specified in bytes. Note that the actual external memory size used is the specified size rounded up to be modulo 64.

Direction: IN Type: UINT32

Default: 64

ulBufferPcmLaw cOCT6100_PCM_U_LAW
cOCT6100_PCM_A_LAW

PCM law of the buffer being loaded into external memory.

Direction: IN Type: UINT32

Default: cOCT6100_PCM_U_LAW

5.6.2 Oct6100BufferPlayoutLoadBlockInit

This function allows the user to initialize loading a buffer into external memory using blocks. The external memory is reserved after this function returns, but the user must call the **Oct6100BufferPlayoutLoadBlock** function to copy the actual buffer into external memory.

Usage

#include "oct6100_api.h"

```
UINT32 Oct6100BufferPlayoutLoadBlockInitDef (
        tPOCT6100_BUFFER_LOAD_BLOCK_INIT    f_pBufferLoadBlockInit );

UINT32 Oct6100BufferPlayoutLoadBlockInit (
        tPOCT6100_INSTANCE_API              f_pApiInstance,
        tPOCT6100_BUFFER_LOAD_BLOCK_INIT    f_pBufferLoadBlockInit );
```

Parameters

f_pApiInstance	Pointer to an instance structure of the chip.
f_pBufferLoadBlockInit	Pointer to a tOCT6100_BUFFER_LOAD_BLOCK_INIT structure. The structure's elements are defined below. The user allocates this structure.

5.6.2.1 tOCT6100_BUFFER_LOAD_BLOCK_INIT Structure

pulBufferIndex 0-1343

Buffer index in the buffer memory. This value is used in future **Oct6100BufferPlayout** function calls to reference this buffer.

Direction: OUT Type: PUINT32

Default: NULL

pulPlayoutFreeMemSize 0 – total space in external memory for playout

Optional parameter that returns the amount of external memory left, in bytes, that can be used for buffer playout after loading the specified buffer. Note that this value does not necessarily refer to a contiguous memory block.

Direction: OUT Type: PUINT32

Default: NULL

ulBufferSize 64 – max space left in external memory

Size of the buffer loaded into external memory. This size is specified in bytes. This value must be modulo 16. Note that the actual external memory size used is the specified size rounded up to be modulo 64.

Direction: IN Type: UINT32

Default: 64

ulBufferPcmLaw cOCT6100_PCM_U_LAW
cOCT6100_PCM_A_LAW

PCM law of the buffer being loaded into external memory.

Direction: IN Type: UINT32

Default: cOCT6100_PCM_U_LAW

5.6.3 Oct6100BufferPlayoutLoadBlock

This function allows the user to load a buffer block into external memory. The user must call the **Oct6100BufferPlayoutLoadBlockInit** function first, to reserve the external memory needed for loading the buffer.

This function must be called repeatedly, loading a portion of the buffer each time. The user can specify the size of each block.

The API does not check that the entire buffer has been loaded. It is the user's responsibility to ensure that all blocks have been loaded.

Usage

```
#include "oct6100_api.h"

UINT32 Oct6100BufferPlayoutLoadBlockDef (
        tPOCT6100_BUFFER_LOAD_BLOCK     f_pBufferLoadBlock );

UINT32 Oct6100BufferPlayoutLoadBlock (
        tPOCT6100_INSTANCE_API          f_pApiInstance,
        tPOCT6100_BUFFER_LOAD_BLOCK     f_pBufferLoadBlock );
```

Parameters

f_pApiInstance	Pointer to an instance structure of the chip.
f_pBufferLoadBlock	Pointer to a tOCT6100_BUFFER_LOAD_BLOCK structure. The structure's elements are defined below. The user allocates this structure.

5.6.3.1 tOCT6100_BUFFER_LOAD_BLOCK Structure

ulBufferIndex 0-1343

Index of the buffer where data should be copied into the chip external memory.

Direction: IN Type: UINT32

Default: cOCT6100_INVALID_VALUE

ulBlockOffset 0 – maximum buffer size

Offset, in bytes, of the first byte in the block to be loaded. This offset is with respect to the beginning of the buffer. This value must be modulo 2.

Direction: IN Type: UINT32

Default: cOCT6100_INVALID_VALUE

ulBlockLength 0 – maximum buffer size

Size of the block to be loaded into external memory. This value must be modulo 2.

Direction: IN Type: UINT32

Default: cOCT6100_INVALID_VALUE

pbyBufferPattern pointer

A byte pointer pointing to a valid buffer to be loaded into the chip's external memory. This is a pointer to the entire buffer. The API uses the **ulBlockOffset** and **ulBlockLength** to index within this buffer and obtain the block to be loaded.

Direction: IN Type: PUINT8

Default: NULL

5.6.4 Oct6100BufferPlayoutUnload

This function allows the user to unload a buffer from the chip's external memory. Note that although unloading a buffer that is currently playing is permitted, the samples played out may be invalid.

Usage

```
#include "oct6100_api.h"

UINT32 Oct6100BufferPlayoutUnloadDef (
            tPOCT6100_BUFFER_UNLOAD          f_pBufferUnload );

UINT32 Oct6100BufferPlayoutUnload (
            tPOCT6100_INSTANCE_API           f_pApiInstance,
            tPOCT6100_BUFFER_UNLOAD          f_pBufferUnload);
```

Parameters

f_pApiInstance	Pointer to an instance structure of the chip.
f_pBufferUnload	Pointer to a tOCT6100_BUFFER_UNLOAD structure. The structure's elements are defined below. The user allocates this structure.

5.6.4.1 tOCT6100_BUFFER_UNLOAD Structure

ulBufferIndex 0-1343

Index of the buffer to be removed from the chip external memory.

Direction: IN Type: UINT32

Default: cOCT6100_INVALID_VALUE

5.6.5 Oct6100BufferPlayoutAdd

This function allows the user to add a buffer to the current list of buffers for one of the channel's output ports.

Usage

```
#include "oct6100_api.h"

UINT32 Oct6100BufferPlayoutAddDef (
            tPOCT6100_BUFFER_PLAYOUT_ADD        f_pBufferPlayoutAdd );

UINT32 Oct6100BufferPlayoutStart (
            tPOCT6100_INSTANCE_API              f_pApiInstance,
            tPOCT6100_BUFFER_PLAYOUT_ADD        f_pBufferPlayoutAdd );
```

Parameters

f_pApiInstance	Pointer to an instance structure of the chip.
f_pBufferPlayoutAdd	Pointer to a tOCT6100_BUFFER_PLAYOUT_ADD structure. The structure's elements are defined below. The user allocates this structure.

5.6.5.1 tOCT6100_BUFFER_PLAYOUT_ADD Structure

ulChannelHndl handle

The handle that identifies the channel on which the specified buffer is to be played. The handle is returned by the call to **Oct6100ChannelOpen**.

Direction: IN Type: UINT32

Default: cOCT6100_INVALID_HANDLE

ulBufferIndex 0-1343

Index of the buffer to be played on the selected port.

Direction: IN Type: UINT32

Default: cOCT6100_INVALID_VALUE

ulPlayoutPort cOCT6100_CHANNEL_PORT_SOUT
cOCT6100_CHANNEL_PORT_ROUT

This is the channel port on which the buffer will be played. Note that the PCM law in which the buffer is played on the ROUT or SOUT port is determined by the PCM law of the RIN and SIN port respectively.

Direction: IN Type: UINT32

Default: cOCT6100_CHANNEL_PORT_ROUT

ulMixingMode cOCT6100_MIXING_MINUS_6_DB
cOCT6100_MIXING_MINUS_12_DB
cOCT6100_MIXING_MUTE

This parameter selects the level of the original signal mixed with the buffer.

Direction: IN Type: UINT32

Default: cOCT6100_MIXING_MINUS_6_DB

lGainDb -24 – 24

This parameter is the gain applied to the selected port's signal during playout.

Direction: IN Type: INT32

Default: 0

fRepeat TRUE / FALSE

This parameter represents whether or not the **ulRepeatCount** parameter should be used for setting the repeat count of the selected buffer. If set to FALSE, the buffer will play only once.

Direction: IN Type: BOOL

Default: FALSE

ulRepeatCount 1 – 32767
cOCT6100_REPEAT_INFINITELY

This parameter represents the number of times that the selected buffer should be played out. If the user sets this parameter to cOCT6100_REPEAT_INFINITELY, once started, this buffer will play until the procedure **Oct6100BufferPlayoutStop** is called. This parameter is ignored if **fRepeat** is set to FALSE.

Direction: IN Type: UINT32

Default: cOCT6100_REPEAT_INFINITELY

ulDuration 32–bit value

This parameter represents the time (duration), in milliseconds, that this buffer should be played. If set, this parameter overrides the **fRepeat** flag.

When this parameter is used, the API converts the **ulDuration** into a **ulRepeatCount** and uses that to configure the device. If **ulRepeatCount** is calculated to be above 32767, then multiple buffer playout events are created to accommodate this.

If the required buffer playout sequence is complicated and requires many different playout events, then using multiple playout events for a single buffer may be undesirable. To get around this problem, a longer buffer can be used, containing many repetitions of the desired signal. This will cause the calculated **ulRepeatCount** to be smaller, thus using less playout events.

Direction: IN Type: UINT32

Default: cOCT6100_INVALID_VALUE

ulBufferLength 64 – size of buffer currently loaded into memory

This parameter allows the user to play only the N first bytes of a buffer. The specified length is with respect to the beginning of the buffer. Setting this value to cOCT6100_AUTO_SELECT will play the buffer completely. This value must be modulo 16.

Direction: IN Type: UINT32

Default: cOCT6100_AUTO_SELECT

5.6.6 Oct6100BufferPlayoutStart

This function allows the user to activate buffer playout on a channel. Note that issuing a playout start command on a port that has no buffers "added" will play nothing, i.e. the signal on the port will be left unchanged.

The playout start command will not return an error if some of the buffers in the list to be played have been unloaded. In this situation, the samples played out may be invalid.

Buffer playout can only be started if the following conditions are met:

- The **fEnableNlp** flag of the channel configuration is set to TRUE.
- The **ulEchoOperationMode** parameter of the channel configuration is not set to cOCT6100_ECHO_OP_MODE_POWER_DOWN or cOCT6100_ECHO_OP_MODE_HT_FREEZE.

Trying to start buffer playout without the above conditions will result in an error from the API.

After a start command is issued, the "added" buffers will begin to play. Once played out, these buffers are removed from the internal buffer list. Therefore, to play the same buffers again on a same channel, they must be "re-added" before another start command is issued.

Usage

```
#include "oct6100_api.h"

UINT32 Oct6100BufferPlayoutStartDef (
            tPOCT6100_BUFFER_PLAYOUT_START      f_pBufferPlayoutStart );

UINT32 Oct6100BufferPlayoutStart (
            tPOCT6100_INSTANCE_API              f_pApiInstance,
            tPOCT6100_BUFFER_PLAYOUT_START      f_pBufferPlayoutStart );
```

Parameters

f_pApiInstance	Pointer to an instance structure of the chip.
f_pBufferPlayoutStart	Pointer to a tOCT6100_BUFFER_PLAYOUT_START structure. The structure's elements are defined below. The user allocates this structure.

5.6.6.1 tOCT6100_BUFFER_PLAYOUT_START Structure

ulChannelHndl handle

The handle that identifies the channel on which buffer playout will be started. The handle is returned by the call to **Oct6100ChannelOpen**.

Direction: IN Type: UINT32

Default: cOCT6100_INVALID_HANDLE

ulPlayoutPort cOCT6100_CHANNEL_PORT_SOUT
cOCT6100_CHANNEL_PORT_ROUT

This is the channel port on which the buffer is to be played. Note that the PCM law in which the buffer is played on the ROUT or SOUT port is determined by the PCM law of the Rin and Sin port respectively.

Direction: IN Type: UINT32

Default: cOCT6100_CHANNEL_PORT_ROUT

fNotifyOnPlayoutStop TRUE / FALSE

This flag indicates whether or not a buffer playout event of type **cOCT6100_BUFFER_PLAYOUT_EVENT_STOP** should be generated when the current list stops playing. The events can be retrieved via a call to **Oct6100BufferPlayoutGetEvent**. This feature is only available if the user allocated a software buffer to store the buffer playout events, when the chip was opened, using the **ulSoftBufferPlayoutEventsBufSize** parameter. Note that issuing a call to **Oct6100BufferPlayoutStop** will not generate an event when the playout stops.

Direction: IN Type: BOOL

Default: FALSE

ulUserEventId 32-bit value

User specified field stored in the API buffer playout structure. This parameter is returned with the channel handle and port when a buffer playout event is detected for the current channel and port. Only used when the **fNotifyOnPlayoutStop** flag is set.

Direction: IN Type: UINT32

Default: cOCT6100_INVALID_VALUE

fAllowStartWhileActive TRUE / FALSE

This flag indicates whether or not the API should return an error to this call if a buffer is currently playing on the specified channel and port. If a buffer is currently playing, and this flag is set to TRUE, the new buffers will be added to the end of the currently playing list.

Direction: IN Type: BOOL

Default: FALSE

5.6.7 Oct6100BufferPlayoutStop

This function stops the buffer playout on the channel's selected port.

Usage

```
#include "oct6100_api.h"

UINT32 Oct6100BufferPlayoutStopDef (
            tPOCT6100_BUFFER_PLAYOUT_STOP        f_pBufferPlayoutStop );

UINT32 Oct6100BufferPlayoutStop (
            tPOCT6100_INSTANCE_API               f_pApiInstance,
            tPOCT6100_BUFFER_PLAYOUT_STOP        f_pBufferPlayoutStop );
```

Parameters

f_pApiInstance	Pointer to an instance structure of the chip.
f_pBufferPlayoutStop	Pointer to a tOCT6100_BUFFER_PLAYOUT_STOP structure. The structure's elements are defined below. The user allocates this structure.

5.6.7.1 tOCT6100_BUF_PLAYOUT_STOP Structure

ulChannelHndl handle

Handle that identifies the echo channel on which the buffer is currently playing. The handle is returned by the call to **Oct6100ChannelOpen**.

Direction: IN Type: UINT32

Default: cOCT6100_INVALID_HANDLE

ulPlayoutPort cOCT6100_CHANNEL_PORT_SOUT
cOCT6100_CHANNEL_PORT_ROUT

This is the echo channel port on which the buffer is currently playing.

Direction: IN Type: UINT32

Default: cOCT6100_CHANNEL_PORT_ROUT

fStopCleanly TRUE / FALSE

This flag indicates if the buffer playing during this function call will stop immediately (FALSE) or if it will stop only once all its samples have been played out (TRUE).

Direction: IN Type: BOOL

Default: TRUE

pfNotifyOnPlayoutStop TRUE / FALSE

Optional output parameter. This variable returns the user configuration for this parameter that was specified when the **Oct6100BufferPlayoutStart** function was called.

Direction: OUT Type: PBOOL

Default: NULL

pfAlreadyStopped TRUE / FALSE

Optional output parameter. This flag will be set to FALSE if a playout list was playing on the selected port before stopping. Note that this parameter will also be set to FALSE if one or more events were added but playout was not started using the **Oct6100BufferPlayoutStart** function.

Direction: OUT Type: PBOOL

Default: NULL

5.7 Caller ID Functions

These functions are used to manage the transmission (using FSK modulation) of Caller ID information. Both ETSI (ETS300 659.1 and 659.2) and Bellcore (GR-30-Core) standards are supported.

The caller ID module must be initialized by invoking the **Oct6100CallerIdInit** function before being used. The standard (ETSI or Bellcore) to be used when generating the messages must be specified using the **ulCallerIdGeneratorType** member of the initialization function.

Two main functions are used to manage the caller ID feature: the transmit and abort functions. The **Oct6100CallerIdTransmit** function is used to start the transmission of a user-specified message. The **Oct6100CallerIdAbort** function is used to stop the transmission of a caller ID. For example, **Oct6100CallerIdAbort** could be invoked if an off-hook signal of the called party was detected.

Another optional function is available to the user: transmission of the Dual-Tone Alerting Signal. The **Oct6100CallerIdTransmitAs** function is used to transmit the terminal alerting signal if doing terminal equipment alerting. This signal is used to signify to the called party that a caller ID message will follow soon. Note that it is the responsibility of the user to detect the terminal equipment acknowledge signal.

The **fEnableCallerId** flag of the tOCT6100_OPEN_CHIP structure must be set to TRUE for caller ID to work.

Note that the caller ID functions are a superset of the buffer playout functions and therefore using caller ID and buffer playout simultaneously on a given channel will not work.

5.7.1 Oct6100CallerIdInit

This function allows the user to initialize the caller ID buffers used to generate the caller ID messages. Note that this function call can take a few milliseconds to complete since all generated buffers must be loaded into the chip's external memory. It is recommended that this function be called before opening any channels to assure the shortest execution time.

Usage

```
#include "oct6100_api.h"

UINT32 Oct6100CallerIdInitDef (
        tPOCT6100_CALLER_ID_INIT        f_pIdInit );

UINT32 Oct6100CallerIdInit (
        tPOCT6100_INSTANCE_API          f_pApiInstance,
        tPOCT6100_CALLER_ID_INIT        f_pIdInit );
```

Parameters

f_pApiInstance	Pointer to an instance structure of the chip.
f_pIdInit	Pointer to a tOCT6100_CALLER_ID_INIT structure. The structure's elements are defined below. The user allocates this structure.

5.7.1.1 tOCT6100_CALLER_ID_INIT Structure

ulCallerIdGeneratorType cOCT6100_CALLER_ID_TYPE_ETSI
cOCT6100_CALLER_ID_TYPE_BELLCORE

Type of generator used in the caller ID module. For the ETSI (ETS300 659.1 and 659.2) standard use cOCT6100_CALLER_ID_TYPE_ETSI. For the Bellcore (GR-30-Core) standard use cOCT6100_CALLER_ID_TYPE_BELLCORE.

Direction: IN Type: UINT32

Default: cOCT6100_CALLER_ID_TYPE_ETSI

5.7.2 Oct6100CallerIdTerminate

This function allows the user to unload the caller ID buffers used to generate the caller ID messages. Note that this function will free all external memory used up by the caller ID module, but will not free the internal associated API instance structures. Upon return of this function, all external memory that was used by the caller ID module will be available for buffer playout.

Usage

```
#include "oct6100_api.h"

UINT32 Oct6100CallerIdTerminateDef (
            tPOCT6100_CALLER_ID_TERMINATE       f_pIdTerminate );

UINT32 Oct6100CallerIdTerminate (
            tPOCT6100_INSTANCE_API              f_pApiInstance,
            tPOCT6100_CALLER_ID_TERMINATE       f_pIdTerminate );
```

Parameters

f_pApiInstance	Pointer to an instance structure of the chip.
f_pIdTerminate	Pointer to a tOCT6100_CALLER_ID_TERMINATE structure. The structure's elements are defined below. The user allocates this structure.

5.7.2.1 tOCT6100_CALLER_ID_TERMINATE Structure

ulDummy 32-bit value

The API does not use this structure member. It exists only to preserve the OCT6100 API functions format.

Direction: IN Type: UINT32

Default: 0

5.7.3 Oct6100CallerIdTransmit

This function allows the user to start transmission of a caller identification message.

Usage

```
#include "oct6100_api.h"

UINT32 Oct6100CallerIdTransmitDef (
            tPOCT6100_CALLER_ID_TRANSMIT        f_pIdTransmit );

UINT32 Oct6100CallerIdTransmit (
            tPOCT6100_INSTANCE_API              f_pApiInstance,
            tPOCT6100_CALLER_ID_TRANSMIT        f_pIdTransmit );
```

Parameters

f_pApiInstance	Pointer to an instance structure of the chip.
f_pIdTransmit	Pointer to a tOCT6100_CALLER_ID_TRANSMIT structure. The structure's elements are defined below. The user allocates this structure.

5.7.3.1 tOCT6100_CALLER_ID_TRANSMIT Structure

byMessageType 8-bit value

Message type to be transmitted. Ignored if **fPreFormattedMessage** is set to TRUE.

Direction: IN Type: UINT8

Default: 0

byMessageLength 0 - 126

Length of the message to be transmitted.

Direction: IN Type: UINT8

Default: 0

pbyMessage pointer

A byte pointer pointing to a valid message to be transmitted.

Direction: IN Type: PUINT8

Default: NULL

ulChannelHndl handle

The handle that identifies the channel on which the caller ID message will be transmitted. The handle is returned by the call to **Oct6100ChannelOpen**. Setting this parameter to cOCT6100_INVALID_HANDLE will return the caller ID message duration without transmitting the message on a channel.

Direction: IN Type: UINT32

Default: cOCT6100_INVALID_HANDLE

ulPlayoutPort cOCT6100_CHANNEL_PORT_SOUT
cOCT6100_CHANNEL_PORT_ROUT
cOCT6100_INVALID_VALUE

This is the channel port on which the caller ID message should be transmitted. Setting this parameter to cOCT6100_INVALID_VALUE will return the caller ID message duration without transmitting the message on a channel port.

Direction: IN Type: UINT32

Default: cOCT6100_CHANNEL_PORT_ROUT

ulNumMarkBits cOCT6100_CALLER_ID_NUM_MARK_BITS_180
cOCT6100_CALLER_ID_NUM_MARK_BITS_80

Number of mark bits that will be inserted after the seizure signal is transmitted. For on-hook caller ID, this should be set to cOCT6100_CALLER_ID_NUM_MARK_BITS_180. For off-hook caller id, this parameter should be set to cOCT6100_CALLER_ID_NUM_MARK_BITS_80.

Direction: IN Type: UINT32

Default: cOCT6100_CALLER_ID_NUM_MARK_BITS_180

fTransmitSeizureSignal TRUE / FALSE

Whether or not the channel seizure signal will be transmitted. For regular on-hook caller ID, this should be TRUE. For off-hook caller ID, this value should be set to FALSE.

Direction: IN Type: BOOL

Default: TRUE

fPreFormattedMessage TRUE / FALSE

If set to TRUE, the message pointed by **pbyMessage** will be considered pre-formatted and no processing will be done on it. The message will be transmitted exactly as it is in the buffer. This mode can be used for single or multiple data messages. The API will assume that the message type, message length and checksum fields are already filled by the user.

Direction: IN Type: BOOL

Default: TRUE

ulPreTransmitDelayMs 0 - 60000

This parameter specifies the amount of pure silence in milliseconds that is inserted before the caller ID information is transmitted. For example, if the ringing time is 2 seconds and the user wishes to wait 500 milliseconds after the ring has stopped before transmitting the caller ID information, this parameter would be set to 2500 ms. Note that a value between 1 and 7 milliseconds will result in 8 milliseconds of delay before transmission.

Direction: IN Type: UINT32

Default: 0

fNotifyOnTransmitEnd TRUE / FALSE

This flag indicates whether or not a buffer playout event of type **cOCT6100_BUFFER_PLAYOUT_EVENT_CALLER_ID_STOP** should be generated when playout of the caller ID message terminates. The events can be retrieved via a call to **Oct6100BufferPlayoutGetEvent**. This feature is only available if the user has allocated a software buffer to store the buffer playout events, when the chip was opened, using the **ulSoftBufferPlayoutEventsBufSize** parameter. Note that issuing a call to **Oct6100CallerIdAbort** will prevent an event from being generated when the transmission aborts.

Direction: IN Type: BOOL

Default: FALSE

ulUserEventId 32-bit value

User-specified field stored in the API buffer playout instance structure. This parameter is returned with the channel handle and port when a buffer playout event is generated for the current channel and port. Only used when the **fNotifyOnTransmitEnd** flag is set.

Direction: IN Type: UINT32

Default: cOCT6100_INVALID_VALUE

pulDurationMs 32-bits value

Optional parameter. The API returns the calculated duration, in milliseconds, of the transmission of the caller identification message. This value does not include the pre-transmission delay inserted using the **ulPreTransmitDelayMs** parameter.

In the case where the duration of a caller identification message is required without actual transmission on a channel port, the **pulDurationMs** parameter should be used with the following parameters:

- **ulChannelHndl** set to cOCT6100_INVALID_HANDLE
- **ulPlayoutPort** set to cOCT6100_INVALID_VALUE

Upon function call return with this set of parameter values, the **pulDurationMs** parameter will contain the duration of the transmission of the configured caller identification message, but no playout activity will occur on a channel port.

Direction: OUT Type: PUINT32

Default: NULL

5.7.4 Oct6100CallerIdTransmitAs

This function allows the user to start transmission of a caller identification Alerting Signal. DT-AS is supported for ETSI and CAS is supported for Bellcore.

Usage

#include "oct6100_api.h"

```
UINT32 Oct6100CallerIdTransmitAsDef (
            tPOCT6100_CALLER_ID_TRANSMIT_AS     f_pIdTransmitAs );

UINT32 Oct6100CallerIdTransmitAs (
            tPOCT6100_INSTANCE_API              f_pApiInstance,
            tPOCT6100_CALLER_ID_TRANSMIT_AS     f_pIdTransmitAs );
```

Parameters

f_pApiInstance	Pointer to an instance structure of the chip.
f_pIdTransmit	Pointer to a tOCT6100_CALLER_ID_TRANSMIT_AS structure. The structure's elements are defined below. The user allocates this structure.

5.7.4.1 tOCT6100_CALLER_ID_TRANSMIT_AS Structure

ulChannelHndl handle

The handle that identifies the channel on which the caller ID alerting signal will be transmitted. The handle is returned by the call to **Oct6100ChannelOpen**.

Direction: IN Type: UINT32

Default: cOCT6100_INVALID_HANDLE

ulPlayoutPort cOCT6100_CHANNEL_PORT_SOUT
cOCT6100_CHANNEL_PORT_ROUT

This is the channel port on which the caller ID alerting signal will be transmitted.

Direction: IN Type: UINT32

Default: cOCT6100_CHANNEL_PORT_ROUT

ulDurationMs cOCT6100_CALLER_ID_AS_DURATION_80_MS
cOCT6100_CALLER_ID_AS_DURATION_100_MS

This parameter specifies the duration, in milliseconds, of the alerting signal. Currently, 2 durations are supported: 80 and 100 milliseconds. For Bellcore type generator, this should be set to 80 milliseconds. For the ETSI on-hook alerting signal, this should be set to 100 milliseconds. For the ETSI off-hook alerting signal, this should be set to 80 milliseconds.

Direction: IN Type: UINT32

Default: 0

ulPreTransmitDelayMs 0 - 60000

This parameter specifies the amount of pure silence, in milliseconds, that is inserted before the caller ID alerting signal is transmitted. For example, if the ringing time is 2 seconds and the user wishes to wait 200 milliseconds after the ring has stopped before transmitting the caller ID alerting signal, this parameter would be set to 2200 ms. Note that a value between 1 and 7 milliseconds will result in 8 milliseconds of delay before transmission.

Direction: IN Type: UINT32

Default: 0

fNotifyOnTransmitEnd TRUE / FALSE

This flag indicates whether or not a buffer playout event of type **cOCT6100_BUFFER_PLAYOUT_EVENT_CALLER_ID_AS_STOP** should be generated when playout of the alerting signal terminates. The events can be retrieved via a call to **Oct6100BufferPlayoutGetEvent**. This feature is only available if the user allocated a software buffer to store the buffer playout events, when the chip was opened, using the **ulSoftBufferPlayoutEventsBufSize** parameter. Note that issuing a call to **Oct6100CallerIdAbort** will prevent and event from being generated when the transmission of the alerting signal aborts.

Direction: IN Type: BOOL

Default: FALSE

ulUserEventId 32-bit value

User specified field stored in the API buffer playout instance structure. This parameter is returned with the channel handle and port when a buffer playout event is detected for the current channel and port. Only used when the **fNotifyOnTransmitEnd** flag is set.

Direction: IN Type: UINT32

Default: cOCT6100_INVALID_VALUE

5.7.5 Oct6100CallerIdAbort

This function allows the user to abort transmission of a caller identification message. This function can be used to stop transmission when the called party goes off-hook.

Note that since caller ID uses buffer playout, any buffer that was playing using the buffer playout functions will also be stopped.

Usage

```
#include "oct6100_api.h"

UINT32 Oct6100CallerIdAbortDef (
            tPOCT6100_CALLER_ID_ABORT       f_pIdAbort );

UINT32 Oct6100CallerIdAbort (
            tPOCT6100_INSTANCE_API          f_pApiInstance,
            tPOCT6100_CALLER_ID_ABORT       f_pIdAbort );
```

Parameters

f_pApiInstance	Pointer to an instance structure of the chip.
f_pIdTransmit	Pointer to a tOCT6100_CALLER_ID_ABORT structure. The structure's elements are defined below. The user allocates this structure.

5.7.5.1 tOCT6100_CALLER_ID_ABORT Structure

ulChannelHndl handle

The handle that identifies the channel on which transmission should be aborted. The handle is returned by the call to **Oct6100ChannelOpen**.

Direction: IN Type: UINT32

Default: cOCT6100_INVALID_HANDLE

ulPlayoutPort cOCT6100_CHANNEL_PORT_SOUT
cOCT6100_CHANNEL_PORT_ROUT

This is the channel port on which the transmission should be aborted.

Direction: IN Type: UINT32

Default: cOCT6100_CHANNEL_PORT_ROUT

pfAlreadyAborted TRUE / FALSE

Optional output parameter. This flag will be set to FALSE if a caller ID message was playing on the selected port before aborting.

Direction: OUT Type: PBOOL

Default: NULL

pfNotifyOnTransmitEnd TRUE / FALSE

Optional output parameter. This parameter returns the user configuration set when the transmission was started.

Direction: OUT Type: PBOOL

Default: NULL

5.8 Event functions

Oct6100EventGet functions are used to retrieve the tone detection events generated in the chip, as well as the buffer playout events.

5.8.1 Oct6100EventGetTone

This function allows the user to retrieve tone detection events. When the appropriate tone detector is enabled, an event is generated when a tone is detected or when the tone stops being detected.

When this function is called, it is possible to receive both the "start" and "stop" event for the same tone. The device generates tone events every 32 ms. If the tone starts and stops in less that 64 ms, then both events will appear in the API queue at the same time.

Usage

```
#include "oct6100_api.h"

UINT32 Oct6100EventGetToneDef (
            tPOCT6100_EVENT_GET_TONE        f_pEventGetTone );

UINT32 Oct6100EventGetTone (
            tPOCT6100_INSTANCE_API          f_pApiInstance,
            tPOCT6100_EVENT_GET_TONE        f_pEventGetTone );
```

Parameters

f_pApiInstance	Pointer to an instance structure of the chip.
f_pEventGetTone	Pointer to a tOCT6100_GET_TONE_EVENT structure. The structure's elements are defined below. The user allocates this structure.

5.8.1.1 tOCT6100_EVENT_GET_TONE Structure

pToneEvent pointer

Pointer to an array of tOCT6100_TONE_EVENT structures. The user must allocate this memory; its size must be consistent with **ulMaxToneEvent**.

Direction: IN/OUT Type: tPOCT6100_TONE_EVENT

Default: NULL

ulMaxToneEvent 1 – **ulSoftToneEventsBufferSize**

Maximum number of tone events that can be returned to the user. The upper range of this parameter is defined by the **ulSoftToneEventsBufSize** parameter of the **tOCT6100_CHIP_OPEN** structure.

Direction: IN Type: UINT32

Default: 1

ulNumValidToneEvent 0 - **ulMaxToneEvent**

The number of tone events returned.

Direction: OUT Type: UINT32

Default: cOCT6100_INVALID_VALUE

fResetBufs TRUE / FALSE

Reset flag for the tone events buffer. Setting this parameter to TRUE will empty the tone events buffer without returning them.

Direction: IN Type: BOOL

Default: FALSE

fMoreEvents TRUE / FALSE

Indicates if more tone events are present in the software buffer.

Direction: OUT Type: BOOL

Default: FALSE

5.8.1.2 tOCT6100_TONE_EVENT Structure

ulChannelHndl handle

Handle of the channel that generated the message.

Direction: OUT Type: UINT32

Default: cOCT6100_INVALID_HANDLE

ulUserChanId 32-bit value

User-defined field associated to the channel that generated this event.

Direction: OUT Type: UINT32

Default: cOCT6100_INVALID_VALUE

ulToneDetected see **Tone Detection Functions Section**

Tone value associated to the message.

Direction: OUT Type: UINT32

Default: cOCT6100_INVALID_VALUE

ulTimestamp 32-bit value

Value of the local timestamp when the message was created. This timestamp is in H.1x0 frame count.

Direction: OUT Type: UINT32

Default: cOCT6100_INVALID_VALUE

ulEventType cOCT6100_TONE_STOP
cOCT6100_TONE_PRESENT

This is the reported event type.

When the tone is detected, the event type reported is cOCT6100_TONE_PRESENT.

If the tone was present but detection stopped, the event type is cOCT6100_TONE_STOP.

Direction: OUT Type: UINT32

Default: cOCT6100_INVALID_VALUE

ulExtToneDetectionPort cOCT6100_CHANNEL_PORT_RIN
cOCT6100_CHANNEL_PORT_SIN
cOCT6100_INVALID_VALUE

Channel port on which the tone was detected. If the channel is not in extended tone detection mode, the API will return a value of cOCT6100_INVALID_VALUE, which can be ignored. This mode is enabled by setting the **fEnableExtToneDetection** parameter to TRUE in the **Oct6100ChipOpen** function call and the **Oct6100ChannelOpen** function.

Direction: OUT Type: UINT32

Default: cOCT6100_INVALID_VALUE

5.8.2 Oct6100BufferPlayoutGetEvent

This function allows the user to retrieve buffer playout events. An event can be generated when a buffer list stops playing.

Usage

#include "oct6100_api.h"

```
UINT32 Oct6100BufferPlayoutGetEventDef (
            tPOCT6100_BUFFER_PLAYOUT_GET_EVENT    f_pBufPlayoutGetEvent);

UINT32 Oct6100BufferPlayoutGetEvent (
            tPOCT6100_INSTANCE_API                f_pApiInstance,
            tPOCT6100_BUFFER_PLAYOUT_GET_EVENT    f_pBufPlayoutGetEvent);
```

Parameters

f_pApiInstance Pointer to an instance structure of the chip.

f_pBufPlayoutGetEvent

Pointer to a tOCT6100_BUFFER_PLAYOUT_GET_EVENT structure. The structure's elements are defined below. The user allocates this structure.

5.8.2.1 tOCT6100_BUFFER_PLAYOUT_GET_EVENT Structure

pBufferPlayoutEvent pointer

Pointer to an array of **tOCT6100_BUFFER_PLAYOUT_EVENT** structures. The user must allocate this memory; its size must be consistent with **ulMaxEvent**. This structure is explained below.

Direction: IN/OUT Type: tPOCT6100_BUFFER_PLAYOUT_EVENT
Default: NULL

ulMaxEvent 1 - **ulSoftBufPlayouEventsBufSize**

Maximum number of buffer playout events that can be returned to the user. The upper range of this parameter is defined by the **ulSoftBufPlayouEventsBufSize** parameter of the **tOCT6100_CHIP_OPEN** structure.

Direction: IN Type: UINT32
Default: 1

ulNumValidEvent 0 - **ulMaxEvent**

The number of valid buffer playout events returned.

Direction: OUT Type: UINT32
Default: cOCT6100_INVALID_VALUE

fResetBufs TRUE / FALSE

Reset flag for the buffer playout events buffer.

Direction: IN Type: BOOL
Default: FALSE

fMoreEvents TRUE / FALSE

Indicates if more buffer playout events are present in the software buffer.

Direction: OUT Type: BOOL
Default: FALSE

5.8.2.2 tOCT6100_BUFFER_PLAYOUT_EVENT Structure

ulChannelHndl handle

Handle of the channel where the event was detected.

Direction: OUT Type: UINT32
Default: cOCT6100_INVALID_HANDLE

ulUserChanId 32-bit value

User-defined field associated to the channel where this event was detected.

Direction: OUT Type: UINT32
Default: cOCT6100_INVALID_VALUE

ulChannelPort cOCT6100_CHANNEL_PORT_SOUT
cOCT6100_CHANNEL_PORT_ROUT

This is the channel port where the event was detected.

Direction: OUT Type: UINT32
Default: cOCT6100_CHANNEL_PORT_ROUT

ulUserEventId 32-bit value

User-defined field supplied when the **Oct6100BufferPlayoutStart**, **Oct6100CallerIdTransmit** or **Oct6100CallerIdTransmitAs** command was issued for the channel and port where the buffer playout event was detected.

Direction: OUT Type: UINT32
Default: cOCT6100_INVALID_VALUE

ulTimestamp 32-bit value

Value of the timestamp, in milliseconds, when the event was created. The precision of this value is dependent on the frequency at which the interrupt service routine is called.

Direction: OUT Type: UINT32
Default: cOCT6100_INVALID_VALUE

ulEventType

cOCT6100_BUFFER_PLAYOUT_EVENT_STOP
cOCT6100_BUFFER_PLAYOUT_EVENT_CALLER_ID_STOP
cOCT6100_BUFFER_PLAYOUT_EVENT_CALLER_ID_AS_STOP

This is the reported buffer playout event type. If a playout buffer currently playing stops, the event type is returned according to the command which started playing it:

- **Oct6100BufferPlayoutStart** will generate an event of type cOCT6100_BUFFER_PLAYOUT_EVENT_STOP.
- **Oct6100CallerIdTransmit** will generate an event of type cOCT6100_BUFFER_PLAYOUT_EVENT_CALLER_ID_STOP.
- **Oct6100CallerIdTransmitAs** will generate an event of type cOCT6100_BUFFER_PLAYOUT_EVENT_CALLER_ID_AS_STOP.

Direction: OUT Type: UINT32

Default: cOCT6100_INVALID_VALUE

5.9 TSI Connection Functions

These functions are used to open and close a TSI connection. A TSI connection reads one TDM sample from a TSST each 125 usec, and outputs it on another one. This functionality is useful for debugging or board tests. TSI connections are not used for normal device operation.

5.9.1 Oct6100TsiCnctOpen

This function opens a connection between two TSSTs.

Usage

```
#include "oct6100_api.h"

UINT32 Oct6100TsiCnctOpenDef (
            tPOCT6100_ TSI_CNCT_OPEN        f_pTsiCnctOpen );

UINT32 Oct6100TsiCnctOpen (
            tPOCT6100_INSTANCE_API          f_pApiInstance,
            tPOCT6100_TSI_CNCT_OPEN         f_pTsiCnctOpen);
```

Parameters

f_pApiInstance	Pointer to an instance structure of the chip.
f_pTsiCnctOpen	Pointer to a tOCT6100_TSI_CNCT_OPEN structure. The structure's elements are defined below. The user allocates this structure.

5.9.1.1 tOCT6100_TSI_CNCT_OPEN Structure

pulTsiCnctHndl handle

The parameter returns the handle for the created TSI connection. This handle is a unique value that identifies the channel in all future function calls affecting this connection. The user allocates the memory for this pointer.

Direction: IN/OUT Type: PUINT32

Default: NULL

ulInputTimeslot

0 – 255 for 16 MHz stream frequency
0 – 127 for 8 MHz stream frequency
0 – 63 for 4 MHz stream frequency
0 – 31 for 2 MHz stream frequency

The timeslot of the TSI connection input TSST. Note that allowed values are affected by the frequency of the clock that controls the **ulInputStream**.

Direction: IN Type: UINT32

Default: cOCT6100_INVALID_TIMESLOT

ulInputStream 0 – 31 for **ulMaxTdmStreams** of 32
0 – 15 for **ulMaxTdmStreams** of 16
0 – 7 for **ulMaxTdmStreams** of 8
0 – 3 for **ulMaxTdmStreams** of 4

The stream of the TSI connection input TSST. Note that allowed values are also affected by the **ulMaxTdmStreams** value specified at the **Oct6100ChipOpen** call.

Direction: IN Type: UINT32

Default: cOCT6100_INVALID_STREAM

ulOutputTimeslot see **ulInputTimeslot** parameter

The timeslot of the TSI connection output TSST. Note that allowed values are affected by the frequency of the clock that controls the **ulOutputStream**.

Direction: IN Type: UINT32

Default: cOCT6100_INVALID_TIMESLOT

ulOutputStream see **ulInputStream** parameter

The stream of the TSI connection output TSST. Note that allowed values are also affected by the **ulMaxTdmStreams** value specified at the **Oct6100ChipOpen** call.

Direction: IN Type: UINT32

Default: cOCT6100_INVALID_STREAM

5.9.2 Oct6100TsiCnctClose

This function closes a TSI connection.

Usage

```
#include "oct6100_api.h"

UINT32 Oct6100TsiCnctCloseDef (
            tPOCT6100_TSI_CNCT_CLOSE        f_pTsiCnctClose );

UINT32 Oct6100TsiCnctClose (
            tPOCT6100_INSTANCE_API          f_pApiInstance,
            tPOCT6100_TSI_CNCT_CLOSE        f_pTsiCnctClose );
```

Parameters

f_pApiInstance	Pointer to an instance structure of the chip.
f_pTsiCnctClose	Pointer to a tOCT6100_TSI_CNCT_CLOSE structure. The structure's elements are defined below. The user allocates this structure.

5.9.2.1 tOCT6100_TSI_CNCT_CLOSE Structure

ulTsiCnctHndl handle

Handle of the TSI connection to be closed. This handle is returned by a call to **Oct6100TsiCnctOpen**. The function sets the handle to an invalid value.

Direction: IN Type: UINT32

Default: cOCT6100_INVALID_HANDLE

5.10 ADPCM Channel Functions

These functions are used to open and close an ADPCM compression or decompression channel. An ADPCM channel reads one TDM sample from a TSST each 125 usec, and outputs the compressed or decompressed result on another one.

5.10.1 Oct6100AdpcmChanOpen

This function opens an ADPCM channel.

Note that such a channel also requires an echo processor.

Usage

```
#include "oct6100_api.h"

UINT32 Oct6100AdpcmChanOpenDef (
            tPOCT6100_ ADPCM_CHAN_OPEN          f_pAdpcmChanOpen );

UINT32 Oct6100AdpcmChanOpen (
            tPOCT6100_INSTANCE_API              f_pApiInstance,
            tPOCT6100_ADPCM_CHAN_OPEN           f_pAdpcmChanOpen );
```

Parameters

f_pApiInstance	Pointer to an instance structure of the chip.
f_pAdpcmChanOpen	Pointer to a tOCT6100_ADPCM_CHAN_OPEN structure. The structure's elements are defined below. The user allocates this structure.

5.10.1.1 tOCT6100_ADPCM_CHAN_OPEN Structure

pulChanHndl handle

This parameter returns the handle for the created ADPCM channel. This handle is a unique value that identifies the channel in all future function calls affecting this channel. The user allocates the memory for this pointer.

Direction: IN/OUT Type: PUINT32

Default: NULL

ulChanMode cOCT6100_ADPCM_ENCODING
cOCT6100_ADPCM_DECODING

Codec configuration of the channel. If set to cOCT6100_ADPCM_ENCODING, the channel will compress PCM samples according to **ulEncodingRate**. If set to cOCT6100_ADPCM_DECODING, the channel will decompressed the input samples according to **ulDecodingRate**.

Direction: IN Type: UINT32

Default: cOCT6100_ADPCM_ENCODING

ulEncodingRate

cOCT6100_G711_64KBPS
cOCT6100_G726_40KBPS
cOCT6100_G726_32KBPS
cOCT6100_G726_24KBPS
cOCT6100_G726_16KBPS
cOCT6100_G727_40KBPS_4_1
cOCT6100_G727_40KBPS_3_2
cOCT6100_G727_40KBPS_2_3
cOCT6100_G727_32KBPS_4_0
cOCT6100_G727_32KBPS_3_1
cOCT6100_G727_32KBPS_2_2
cOCT6100_G727_24KBPS_3_0
cOCT6100_G727_24KBPS_2_1
cOCT6100_G727_16KBPS_2_0

This parameter represents the rate of the encoder. G.727 defines contain a suffix: The first number is the number of core bits, and the second is the number of enhanced bits

The API will ignore this parameter if **ulChanMode** is set to cOCT6100_ADPCM_DECODING.

Direction: IN Type: UINT32

Default: cOCT6100_G726_32KBPS

ulDecodingRate

cOCT6100_G711_64KBPS
cOCT6100_G726_40KBPS
cOCT6100_G726_32KBPS
cOCT6100_G726_24KBPS
cOCT6100_G726_16KBPS
cOCT6100_G727_2C_ENCODED
cOCT6100_G727_3C_ENCODED
cOCT6100_G727_4C_ENCODED
cOCT6100_G726_ENCODED
cOCT6100_G711_G726_ENCODED
cOCT6100_G711_G727_2C_ENCODED
cOCT6100_G711_G727_3C_ENCODED
cOCT6100_G711_G727_4C_ENCODED

The API will ignore this parameter if **ulChanMode** is set to cOCT6100_ADPCM_ENCODING.

This parameter represents the rate of the decoder. G.727 defines contain a suffix: The first number is the number of core bits, and the second is the number of enhanced bits

If the decoding rate is a combination of G.711 with either G.726 or G.727, the number of TSSTs assigned to the Decoder input port must be set to 2.

Direction: IN Type: UINT32

Default: cOCT6100_G726_32KBPS

ulInputTimeslot 0 – 255 for 16 MHz stream frequency
0 – 127 for 8 MHz stream frequency
0 – 63 for 4 MHz stream frequency
0 – 31 for 2 MHz stream frequency

The timeslot of the channel input TSST. Note that allowed values are affected by the frequency of the clock that controls the **ulInputStream**.

Direction: IN Type: UINT32

Default: cOCT6100_INVALID_TIMESLOT

ulInputStream 0 – 31 for **ulMaxTdmStreams** of 32
0 – 15 for **ulMaxTdmStreams** of 16
0 – 7 for **ulMaxTdmStreams** of 8
0 – 3 for **ulMaxTdmStreams** of 4

The stream of the channel input TSST. Note that allowed values are also affected by the **ulMaxTdmStreams** value specified at the **Oct6100ChipOpen** call.

Direction: IN Type: UINT32

Default: cOCT6100_INVALID_STREAM

ulInputNumTssts 1, 2

This parameter indicates the number of TSSTs used to read the input samples. See the **TSST Formats section** for more information.

Direction: IN Type: UINT32

Default: 1

ulInputPcmLaw cOCT6100_PCM_U_LAW
cOCT6100_PCM_A_LAW

PCM law of the samples read from the H.100 bus.

Direction: IN Type: UINT32

Default: cOCT6100_PCM_U_LAW

ulOutputTimeslot see **ulInputTimeslot** parameter

The timeslot of the channel output TSST. Note that allowed values are affected by the frequency of the clock that controls the **ulOutputStream**.

Direction: IN Type: UINT32

Default: cOCT6100_INVALID_TIMESLOT

ulOutputStream see **ulInputStream** parameter

The stream of the channel output TSST. Note that allowed values are also affected by the **ulMaxTdmStreams** value specified at the **Oct6100ChipOpen** call.

Direction: IN Type: UINT32

Default: cOCT6100_INVALID_STREAM

ulOutputNumTssts 1, 2

This parameter indicates the number of TSSTs used to drive the output samples. See the **TSST Formats section** for more information.

Direction: IN Type: UINT32

Default: 1

ulOutputPcmLaw cOCT6100_PCM_U_LAW
cOCT6100_PCM_A_LAW

PCM law of the samples written to the H.100 bus.

Direction: IN Type: UINT32

Default: cOCT6100_PCM_U_LAW

ulAdpcmNibblePosition cOCT6100_ADPCM_IN_LOW_BITS
cOCT6100_ADPCM_IN_HIGH_BITS

This is the position of the ADPCM bits within the H.100 TDM timeslot.

Direction: IN Type: UINT32

Default: cOCT6100_ADPCM_IN_LOW_BITS

5.10.2 Oct6100AdpcmChanClose

This function closes an ADPCM channel.

Usage

```
#include "oct6100_api.h"

UINT32 Oct6100AdpcmChanCloseDef (
            tPOCT6100_ADPCM_CHAN_CLOSE      f_pAdpcmChanClose );

UINT32 Oct6100AdpcmChanClose (
            tPOCT6100_INSTANCE_API          f_pApiInstance,
            tPOCT6100_ADPCM_CHAN_CLOSE      f_pAdpcmChanClose );
```

Parameters

f_pApiInstance	Pointer to an instance structure of the chip.
f_pAdpcmChanClose	Pointer to a tOCT6100_ADPCM_CHAN_CLOSE structure. The structure's elements are defined below. The user allocates this structure.

5.10.2.1 tOCT6100_ADPCM_CHAN_CLOSE Structure

ulChanHndl handle

Handle of the ADPCM channel to be closed. This handle is returned by a call to **Oct6100AdpcmChanOpen**. The function sets the handle to an invalid value.

Direction: IN Type: UINT32

Default: cOCT6100_INVALID_HANDLE

5.11 Interrupt Functions

Refer to the **Target System Architecture** description for the interrupt treatment flow.

The interrupts are divided into the following categories:

Fatal – Indicates that the chip has encountered a fatal error, and must be reset to operate properly.

Error – Indicates that the chip has detected an error that must be handled by the user application. There is no recovery required by the chip, the severity and/or recovery, if any, can only be determined by the application.

API Sync – This interrupt is used by the API to maintain synchronization with the chip. The API schedules interrupts at regular intervals to cause the ISR to be called. This is done to prevent corruption in the device statistics and counters caused by counters wrapping.

The category to which an interrupt belongs to is indicated by the name's prefix of interrupt.

5.11.1 Oct6100InterruptServiceRoutine

This function is called by the OS Interrupt Service Routine (ISR) to service the interrupts. It takes the appropriate action to treat any active interrupts when called by the OS ISR.

The user can enable interrupts using the **Oct6100ChipOpen** or **Oct6100InterruptConfigure** function. Disabled interrupts are not serviced by this routine and will not generate a hardware interrupt on the interrupt pin of the device.

To create a polled system, the user must call this routine often enough for proper operation of the device. In systems that use tone detection and/or buffer playout events, the polling frequency should be fast enough to empty the hardware event buffers and receive the event information in a timely manner. A 20 ms polling interval should suffice most applications where events are required.

In systems that do not make use of events, polling the device at an interval of 20 seconds should suffice.

This function will reset all conditions causing the interrupt. This ensures that the interrupt pin will be inactive when it returns from the function call.

Usage

```
#include "oct6100_api.h"

void Oct6100InterruptServiceRoutineDef (
        tPOCT6100_INTERRUPT_FLAGS       f_pInterruptFlags );

void Oct6100InterruptServiceRoutine (
        tPOCT6100_INSTANCE_API          f_pApiInstance,
        tPOCT6100_INTERRUPT_FLAGS       f_pInterruptFlags );
```

Parameters

f_pApiInstance	Pointer to the tOCT6100_INSTANCE_API structure of the chip to be serviced.
f_pInterruptFlags	Pointer to a tOCT6100_INTERRUPT_FLAGS structure. This structure indicates which errors or alarms were detected or treated by the **Oct6100InterruptServiceRoutine** function.

5.11.1.1 tOCT6100_INTERRUPT_FLAGS Structure

The following parameters indicate the events detected during the ISR operation.

fFatalGeneral TRUE / FALSE

If TRUE, an internal fatal chip error has been detected.

Direction: OUT Type: BOOL

Default: FALSE

ulFatalGeneralFlags 32-bit value

If **fFatalGeneral** is set to TRUE, this mask contains the type of fatal general error(s) detected. Please report this information to Octasic. The mask can be composed of the following error types:

- cOCT6100_FATAL_GENERAL_ERROR_TYPE_1
- cOCT6100_FATAL_GENERAL_ERROR_TYPE_2
- cOCT6100_FATAL_GENERAL_ERROR_TYPE_3
- cOCT6100_FATAL_GENERAL_ERROR_TYPE_4
- cOCT6100_FATAL_GENERAL_ERROR_TYPE_5

Direction: OUT Type: UINT32

Default: 0

fFatalReadTimeout TRUE / FALSE

If TRUE, a read to the external memory has failed.

Direction: OUT Type: BOOL

Default: FALSE

fErrorRefreshTooLate TRUE / FALSE

If TRUE, refreshes to the external memory may have exceeded the configured period. Information in the external memory bank may be corrupt.

Direction: OUT Type: BOOL

Default: FALSE

fErrorPllJitter TRUE / FALSE

If TRUE, the chip read invalid data from the external memory. Information in the external memory bank may have been corrupted. Contact Octasic if this error persists.

Direction: OUT Type: BOOL

Default: FALSE

fErrorOverflowToneEvents TRUE / FALSE

If TRUE, the tone event buffer located in external memory has overflowed.

Direction: OUT Type: BOOL

Default: FALSE

fErrorH100OutOfSync TRUE / FALSE

If TRUE, the H.100 slave has lost its framing on the bus, causing the chip's H.100 data pins to be tri-stated.

Direction: OUT Type: BOOL

Default: FALSE

fErrorH100ClkA TRUE / FALSE

If TRUE, the CT_C8_A clock behavior does not conform to the H.100 specification.

Direction: OUT Type: BOOL

Default: FALSE

fErrorH100FrameA TRUE / FALSE

If TRUE, the CT_FRAME_A clock behavior does not conform to the H.100 specification.

Direction: OUT Type: BOOL

Default: FALSE

fErrorH100ClkB TRUE / FALSE

If TRUE, the CT_C8_B clock is not running at 16.384 MHz. This parameter is valid only if the tOCT6100_CHIP_OPEN parameter **fEnableFastH100Mode** is set to TRUE

Direction: OUT Type: BOOL

Default: FALSE

fToneEventsPending TRUE / FALSE

If TRUE, tone events are present within the tone event software buffer. Use function **Oct6100EventGetTone** to retrieve these events. Note that tone events do not generate a hardware interrupt in an interrupt-driven system. The user should poll the ISR regularly and rapidly enough to reach the desired tone detection responsiveness.

Direction: OUT Type: BOOL

Default: FALSE

fBufferPlayoutEventsPending TRUE / FALSE

If TRUE, buffer playout events are present within the buffer playout event software buffer. Use function **Oct6100BufferPlayoutGetEvent** to retrieve these events. Note that buffer playout events do not generate a hardware interrupt in an interrupt-driven system. The user should poll the ISR regularly and rapidly enough to reach the desired buffer playout event detection responsiveness.

Direction: OUT Type: BOOL

Default: FALSE

fApiSynch TRUE / FALSE

If TRUE, the chip interrupted the API for purposes of maintaining synchronization with the API. This is used for information purposes only.

Direction: OUT Type: BOOL

Default: FALSE

5.11.2 Oct6100InterruptMask

The operating system's interrupt service routine uses this function to disable the chip's interrupt pin. When the chip generates an interrupt, the OS starts its interrupt service routine (see the **System Architecture** description in the **Overview** section).

The API's ISR must be called to treat the interrupt. Either the OS calls the API's ISR directly from its ISR, or it defers the treatment of the ISR to a later time, and at a lower CPU priority level. In this case, the interrupt pin of the chip must be disabled until the current interrupt has been treated. This function serves this purpose.

The function first reads the chip's interrupt register to determine the source of the interrupt (many devices can share the same interrupt line). If the chip is the source of the interrupt, the function performs a single write to the chip's interrupt register, which disables the interrupt pins from generating another interrupt for up to 60 ms.

After the disable timer has expired, the interrupt pin will be reactivated. If the conditions causing the original interrupt persist or a new event has occurred, the interrupt pin will be asserted right after the disable timer expires. The API's ISR will re-enable the interrupt pin when it completes allowing new interrupts to occur in potentially less than 60 ms.

Usage

```
#include "oct6100_apimi.h"

UINT32 Oct6100InterruptMaskDef (
            tPOCT6100_INTERRUPT _MASK       f_pInterruptMask );

UINT32 Oct6100InterruptMask (
            tPOCT6100_INTERRUPT _MASK       f_pInterruptMask );
```

Parameters

f_pInterruptMask — Pointer to a tOCT6100_INTERRUPT_MASK structure. The structure's elements are defined below.

5.11.2.1 tOCT6100_INTERRUPT_MASK Structure

ulUserChipIndex identifier

This parameter identifies the chip instance that is targeted. It corresponds to the **ulUserChipId** parameter of the **Oct6100ChipOpen** function. See the **System Architecture** description in the **Overview** section for more details.

Direction: IN Type: UINT32

Default: cOCT6100_INVALID_CHIP_NUMBER

5.11.3 Oct6100InterruptConfigure

This function is used to change the current configuration of the interrupt servicing. Note that the chip should be successfully opened before calling this function.

Usage

```
#include "oct6100_api.h"

void Oct6100InterruptConfigureDef (
        tPOCT6100_INTERRUPT_CONFIGURE    f_pInterruptConfigure );

void Oct6100InterruptConfigure (
        tPOCT6100_INSTANCE_API           f_pApiInstance,
        tPOCT6100_INTERRUPT_CONFIGURE    f_pInterruptConfigure );
```

Parameters

f_pApiInstance — Pointer to the tOCT6100_INSTANCE_API structure of the chip for which the interrupts are to be reconfigured.

f_pInterruptConfigure — Pointer to an interrupt configuration structure. The structure's elements are defined below.

5.11.3.1 tOCT6100_INTERRUPT_CONFIGURE Structure

The following parameters determine which events will trigger an interrupt, and how that event will be treated by the API's ISR. The description of the interrupts is provided in the description of the **tOCT6100_INTERRUPT_FLAGS** structure.

Interrupts can be permanently disabled, preventing them from asserting the hardware interrupt pin and generating an event (**cOCT6100_INTERRUPT_DISABLE**).

If the interrupt is enabled, it can operate in one of two modes:

- It can be reset and kept enabled (**cOCT6100_INTERRUPT_NO_TIMEOUT**) allowing another interrupt of the same type to be detected immediately.
- It can be cleared and disabled for a certain period (**cOCT6100_INTERRUPT_TIMEOUT**), masking the interrupt for a configurable amount of time. This can be used to prevent being flooded by less important interrupts.

ulFatalGeneralConfig — cOCT6100_INTERRUPT_DISABLE, cOCT6100_INTERRUPT_NO_TIMEOUT

The configuration of the general fatal interrupt. The interrupt can be prevented from asserting the hardware interrupt pin and generating an event (cOCT6100_INTERRUPT_DISABLE). If the interrupt is enabled, it can be reset and kept enabled (cOCT6100_INTERRUPT_NO_TIMEOUT). The configuration of this interrupt can be changed dynamically, see **Oct6100InterruptConfigure**.

Default: cOCT6100_INTERRUPT_NO_TIMEOUT

ulFatalMemoryConfig cOCT6100_INTERRUPT_DISABLE
cOCT6100_INTERRUPT_NO_TIMEOUT
cOCT6100_INTERRUPT_TIMEOUT

The configuration of all fatal interrupts associated to the operation of the external memories. The interrupt can be disabled from asserting the hardware interrupt pin and generating an event (cOCT6100_INTERRUPT_DISABLE). If the interrupt is enabled, it can behave in one of two ways once the interrupt has been treated. It can be reset and kept enabled (cOCT6100_INTERRUPT_NO_TIMEOUT) or it can be cleared and disabled for a timeout period time (cOCT6100_INTERRUPT_TIMEOUT). In the latter case, the timeout period is specified by the ulFatalMemoryTimeout parameter. The configuration of this interrupt can be changed dynamically, see **Oct6100InterruptConfigure**. This parameter indicates the operation of the following members of the tOCT6100_INTERRUPT_FLAGS structure:

fFatalReadTimeout

Default: cOCT6100_INTERRUPT_NO_TIMEOUT

ulErrorMemoryConfig see **ulFatalMemoryConfig**

The configuration of all data integrity interrupts associated to the operation of the external memories. This parameter indicates the operation of the following members of the tOCT6100_INTERRUPT_FLAGS structure:

fErrorRefreshTooLate

fErrorPllJitter

Default: cOCT6100_INTERRUPT_NO_TIMEOUT

ulErrorOverflowToneEventsConfig see **ulFatalMemoryConfig**

The configuration of all error interrupts associated to the overflow of the tone events buffer contained in external memory. This parameter indicates the operation of the following members of the tOCT6100_INTERRUPT_FLAGS structure:

fErrorOverflowToneEvents

Default: cOCT6100_INTERRUPT_NO_TIMEOUT

ulErrorH100Config see **ulFatalMemoryConfig**

The configuration of all error interrupts associated to H.100 bus. This parameter indicates the operation of the following members of the tOCT6100_INTERRUPT_FLAGS structure:

fErrorH100OutOfSync
fErrorH100ClkA
fErrorH100ClkB
fErrorH100FrameA

Default: cOCT6100_INTERRUPT_NO_TIMEOUT

ulFatalMemoryTimeout 10 – 10000 ms

This parameter specifies the timeout period of all the fatal memory interrupts when the **ulFatalMemoryConfig** parameter specifies cOCT6100_INTERRUPT_TIMEOUT. This parameter is rounded up to the next nearest multiple of 10 ms before being applied.

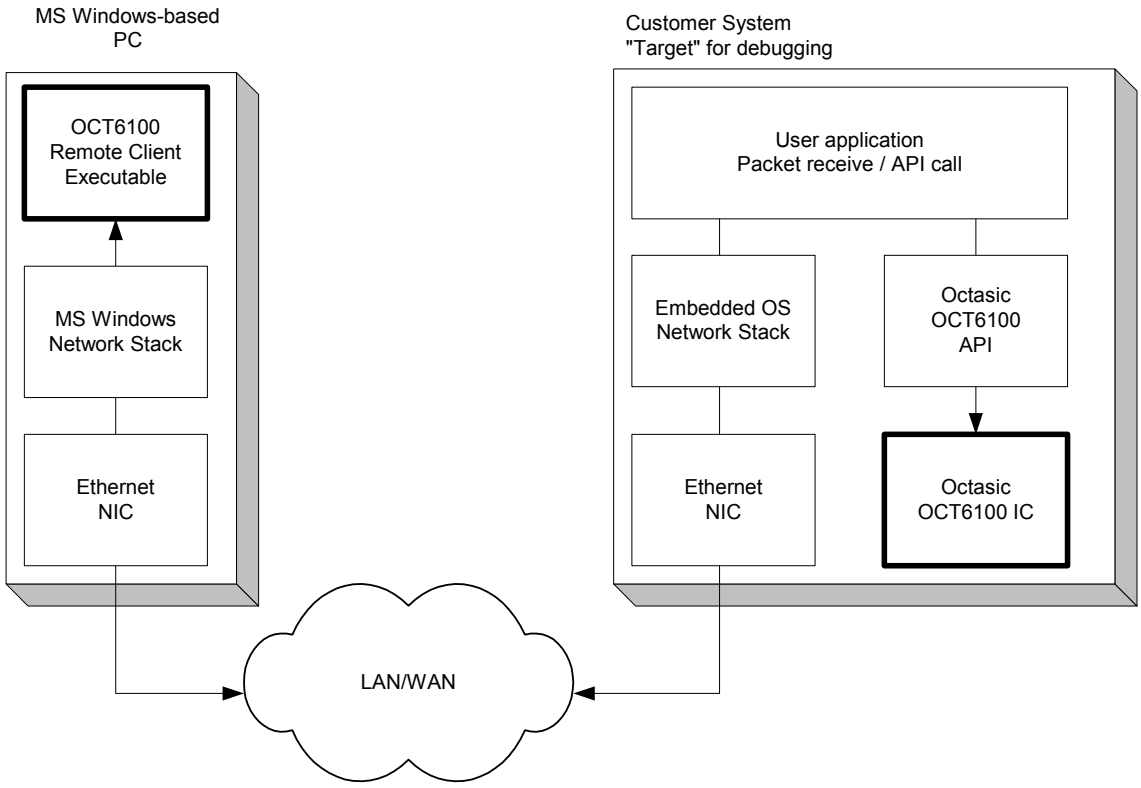
Direction: IN Type: UINT32

Default: 100

ulErrorMemoryTimeout	see **ulFatalMemoryTimeout**
ulErrorOverflowToneEventsTimeout	see **ulFatalMemoryTimeout**
ulErrorH100Timeout	see **ulFatalMemoryTimeout**

5.12 Remote Debugging

The API supports remote debugging. This allows the customer or an Octasic support engineer to run a debugger executable remotely. This executable is provided by Octasic and is called the **Oct6100 Remote Client**. This approach allows the user to monitor the device and to perform real-time captures of the voice streams and captures of the device's internal state. Communication between the debugger executable and the API is done via a non-guaranteed packet based messaging protocol using IP and UDP.



When the OCT6100 Remote Client is launched, a destination IP address and UDP port are selected. A socket is opened in the customer's system to accept these packets. When packets are received with the agreed source and destination addresses and ports, the packet must be passed to the API. The API only needs the packet's payload. Thus, the user must provide code to open a Socket, receive the Remote Client's packets, strip off the IP and UDP headers, and pass the payload to the API. The packet payload is passed to the API via the **Oct6100RemoteDebug** function. Once the API call completes, it will return a "response" packet. It is the user's responsibility to send this packet back to the Oct6100 Remote Client.

In the case where the target platform does not support IP/UDP, software can be written by the user to convert the remote debugger's packets to another messaging protocol supported by the target platform. For example, in a CompactPCI environment, the main CPU board can convert packets received on its Ethernet port to messages passed onto another CompactPCI board on which the target platform resides. This can be over PCI for example. An application must be written by the user to open a target socket on the CPU board and to send/receive debug messages to the target platform. The messaging protocol between the CPU board and the target platform is design-specific. Octasic can provide assistance with the definition of such an interface.

5.12.1 Oct6100RemoteDebug

This function interprets the remote debugging packets received by the user's software. Commands contained in the packet are executed by the API. In addition, a response packet is constructed and returned by the function. It is the responsibility of the user's software to transmit the response packet back to the source of the debugging packet.

Usage

```
#include "oct6100_api.h"

void Oct6100RemoteDebugDef (
        tPOCT6100_REMOTE_DEBUG          f_pRemoteDebug );

void Oct6100RemoteDebug (
        tPOCT6100_INSTANCE_API          f_pApiInstance,
        tPOCT6100_REMOTE_DEBUG          f_pRemoteDebug );
```

Parameters

f_pApiInstance	pointer to the tOCT6100_INSTANCE_API structure of the chip for which the interrupts are to be reconfigured.
f_pRemoteDebug	pointer to a tOCT6100_REMOTE_DEBUG structure. The definitions of the structure's elements are listed below.

5.12.1.1 Structure tOCT6100_REMOTE_DEBUG

ulReceivedPktLength 32 – 32768

The length of the received packet, in bytes. The length must be a multiple of 4.

Direction: IN Type: UINT32

Default: 0

pulReceivedPktPayload pointer to array

The payload of the received packet. The payload is contained in an array of UINT32s. The data is placed in the array as follows:

```
                                31 24  23 16  15  8  7   0
pulReceivedPktPayload[ 0 ] =  Byte 0  Byte 1  Byte 2  Byte 3
pulReceivedPktPayload[ 1 ] =  Byte 4  Byte 5  Byte 6  Byte 7
                    ……..
```

Where Byte X is the Xth received byte of the packet's payload.

Direction: IN/IN Type: PUINT32

Default: NULL

ulMaxResponsePktLength 32 – 32768

The size of the buffer provided to contain the response packet payload constructed by the API, **pulResponsePktPayload**, in bytes. Must always be equal or greater than **ulReceivedPktLength**.

Direction: IN Type: UINT32

Default: 0

ulResponsePktLength 0 – 32768

The actual size of the response packet payload constructed by the API, contained in the user provided buffer **pulResponsePktPayload**, in bytes.

Direction: OUT Type: UINT32

Default: 0

pulResponsePktPayload pointer to array

The payload of the API constructed response packet. The payload is contained in an array of UINT32s. The data is placed in the array as follows:

	31 24	23 16	15 8	7 0
pulResponsePktPayload[0] =	Byte 0	Byte 1	Byte 2	Byte 3
pulResponsePktPayload[1] =	Byte 4	Byte 5	Byte 6	Byte 7

…….

Where Byte X is the Xth transmitted byte of the packet's payload.

Direction: IN/OUT Type: PUINT32

Default: NULL

5.13 Monitoring Functions

To make board-level integration and debugging easier, the OCT6100 API offers monitoring functions allowing the user to retrieve live debug information specific to a channel from the chip. This live information can then be stored into a binary file and sent to Octasic Support.

The **fEnableChannelRecording** parameter of the **Oct6100ChipOpen** function must be set to TRUE. In the case of 672 channel devices, 1 channel will be used for this function.

The first thing to be done when debugging a channel is to select the debug channel using the **Oct6100DebugSelectChannel** function. This will tell the chip to start monitoring the channel.

Once the channel is selected, the user can dump the recorded information using the **Oct6100DebugGetData** function. This dump contains, among other things, up to 2 minutes of recorded PCM signal from all ports of the channel.

In case of a problem with the performance of the echo canceller, the host application should write the retrieved data to a binary file and send it to Octasic Support to have the issue resolved.

All these monitoring functions can be used dynamically on any channel.

NOTE: The selected channel's **ulEchoOperationMode** parameter must be different from cOCT6100_ECHO_OP_MODE_POWER_DOWN for monitored information to be valid.

5.13.1 Oct6100EnableChannelRecording

The function enables recording on the debug channel. The function returns an error if the debug channel is currently active. The channel cannot be used for echo cancelation while recording is enabled. The channel can be used again for echo cancelation if recording is disabled.

Usage

```
#include "oct6100_api.h"

UINT32 Oct6100EnableChannelRecordingDef (
            tPOCT6100_ENABLE_CHANNEL_RECORDING    f_pChannelRec );

UINT32 Oct6100EnableChannelRecording (
            tPOCT6100_INSTANCE_API                f_pApiInstance,
            tPOCT6100_ENABLE_CHANNEL_RECORDING    f_ pChannelRec );
```

Parameters

f_pApiInstance	Pointer to an instance structure of the chip.
f_pChannelRec	Pointer to a tOCT6100_ENABLE_CHANNEL_RECORDING structure. The structure's elements are defined below. The user allocates this structure.

5.13.1.1 tOCT6100_ENABLE_CHANNEL_RECORDING Structure

pulChannelHndlConflict handle

Contains the handle of the debug channel if it's currently active. This channel must not be active to enable recording.

Direction: OUT Type: UINT32

Default: cOCT6100_INVALID_HANDLE

5.13.2 Oct6100DisableChannelRecording

The function disables recording on the debug channel.

Usage

```
#include "oct6100_api.h"

UINT32 Oct6100DisableChannelRecordingDef (
            tPOCT6100_DISABLE_CHANNEL_RECORDING   f_pChannelRec );

UINT32 Oct6100DisableChannelRecording (
            tPOCT6100_INSTANCE_API                f_pApiInstance,
            tPOCT6100_DISABLE_CHANNEL_RECORDING   f_ pChannelRec );
```

Parameters

f_pApiInstance	Pointer to an instance structure of the chip.
f_pChannelRec	Pointer to a tOCT6100_DISABLE_CHANNEL_RECORDING structure. The structure's elements are defined below. The user allocates this structure.

5.13.2.1 tOCT6100_DISABLE_CHANNEL_RECORDING Structure

ulUnused 32-bit value

Place holder. There are no parameters for this function as this time.

Direction: OUT Type: UINT32

Default: 0

5.13.3 Oct6100DebugSelectChannel

This function selects the debug channel to be monitored by the chip.

Usage

```
#include "oct6100_api.h"

void Oct6100DebugSelectChannelDef (
        tPOCT6100_DEBUG_SELECT_CHANNEL      f_pDebugSelectChan );

void Oct6100DebugSelectChannel (
        tPOCT6100_INSTANCE_API              f_pApiInstance,
        tPOCT6100_DEBUG_SELECT_CHANNEL      f_pDebugSelectChan );
```

Parameters

f_pApiInstance	pointer to the tOCT6100_INSTANCE_API structure of the chip for which the interrupts are to be reconfigured.
f_pDebugSelectChan	pointer to a tOCT6100_DEBUG_SELECT_CHANNEL structure. The definitions of the structure's elements are listed below.

5.13.3.1 Structure tOCT6100_ DEBUG_SELECT_CHANNEL

ulChannelHndl handle

This is the handle of the channel to be monitored. This value was returned by a call to Oct6100ChannelOpen. To stop the monitoring process, set this parameter to cOCT6100_INVALID_HANDLE.

Direction: IN Type: UINT32

Default: cOCT6100_INVALID_HANDLE

5.13.4 Oct6100DebugGetData

This function retrieves the recorded data of the debug channel. This function is called repeatedly until all bytes have been retrieved by the API and written to a file by the user. Refer to the sample code after the parameter description for an example on how to use the debug functions.

When this function is first called, the API will instruct the OCT6100 to freeze the recording process, so that nothing is lost. The user must then call this function repeatedly to obtain the debug data. Once all the recorded data has been retrieved, the chip resumes recording data.

Usage

```
#include "oct6100_api.h"

void Oct6100DebugGetDataDef (
            tPOCT6100_DEBUG_GET_DATA_       f_pGetData );

void Oct6100DebugGetData (
            tPOCT6100_INSTANCE_API          f_pApiInstance,
            tPOCT6100_DEBUG_GET_DATA        f_pGetData );
```

Parameters

f_pApiInstance	pointer to the tOCT6100_INSTANCE_API structure of the chip for which the interrupts are to be reconfigured.
f_pGetData	pointer to a tOCT6100_DEBUG_GET_DATA structure. The definitions of the structure's elements are listed below.

5.13.4.1 Structure tOCT6100_ DEBUG_GET_DATA

ulGetDataMode

cOCT6100_DEBUG_GET_DATA_MODE_120S
cOCT6100_DEBUG_GET_DATA_MODE_120S_LITE
cOCT6100_DEBUG_GET_DATA_MODE_16S
cOCT6100_DEBUG_GET_DATA_MODE_16S_LITE
cOCT6100_DEBUG_GET_DATA_MODE_CORE_DUMP

This parameter determines how much recorded data should be retrieved. The x120S modes will retrieve the last 2 minutes of data and the x16S modes will retrieve the last 16 seconds of data. The "CORE DUMP" mode retrieves the maximum recorded data (2 minutes or 16 seconds depending on the loaded image) as well as a full dump of the internal and external memory structures. The "LITE" modes should always be used unless otherwise specified by a support agent.

The error cOCT6100_ERR_NOT_SUPPORTED_DEBUG_DATA_MODE_120S will be returned if the firmware does not support the requested mode.

For a given dump, the user cannot change the value of this parameter. Once the mode has been selected, the entire dump must be completed at that length.

The following table summarizes the resulting size of the dumps according to the selected data mode.

Parameter value	Approximate dump size
cOCT6100_DEBUG_GET_DATA_MODE_120S	4274 KB
cOCT6100_DEBUG_GET_DATA_MODE_120S_LITE	3239 KB
cOCT6100_DEBUG_GET_DATA_MODE_16S	1376 KB
cOCT6100_DEBUG_GET_DATA_MODE_16S_LITE	407 KB
cOCT6100_DEBUG_GET_DATA_MODE_CORE_DUMP	up to 128 MB

These are maximum values, applicable when **ulGetDataContent** is set to cOCT6100_DEBUG_GET_DATA_CONTENT_COMPLETE.

Direction: IN Type: UINT32

Default: cOCT6100_DEBUG_GET_DATA_MODE_120S_LITE

ulGetDataContent cOCT6100_DEBUG_GET_DATA_CONTENT_COMPLETE
cOCT6100_DEBUG_GET_DATA_CONTENT_RIN_PCM
cOCT6100_DEBUG_GET_DATA_CONTENT_SIN_PCM
cOCT6100_DEBUG_GET_DATA_CONTENT_SOUT_PCM

This parameter determines the content of the debug data to be retrieved. The default value, cOCT6100_DEBUG_GET_DATA_CONTENT_COMPLETE, should be used to retrieve all recorded information related to the debug channel.

Specifying cOCT6100_DEBUG_GET_DATA_CONTENT_*xxx*_PCM will instruct the API to only return the PCM recorded data of the selected port. In this case, the data is a direct 8-bit PCM recording of the TDM H.1x0 voice stream as seen or written on the selected port. The PCM data is encoded in the channel's current law. The cOCT6100_DEBUG_GET_DATA_CONTENT_*xxx*_PCM contents can be used to retrieve the PCM data of all ports one after the other.

Note that if one of the cOCT6100_DEBUG_GET_DATA_CONTENT_*xxx*_PCM contents are used, the **Oct6100DebugSelectChannel** function should be called again once the dumps are finished to instruct the API to continue recording. Conversely, using cOCT6100_DEBUG_GET_DATA_CONTENT_COMPLETE causes the API to resume recording once the data has been retrieved.

Finally, if the data mode is set to cOCT6100_DEBUG_GET_DATA_MODE_CORE_DUMP, the parameter can only be set to cOCT6100_DEBUG_GET_DATA_CONTENT_COMPLETE. The API will report an error for all other cases of **ulGetDataContent** when **ulGetDataMode** is set to cOCT6100_DEBUG_GET_DATA_MODE_CORE_DUMP.

Direction: IN Type: UINT32

Default: cOCT6100_DEBUG_GET_DATA_CONTENT_COMPLETE

ulRemainingNumBytes 32-bit value

How many bytes are left to be retrieved before the dump is complete. The **Oct6100DebugGetData** function should be called in a loop until this parameter is equal to 0.

Direction: OUT Type: UINT32

Default: cOCT6100_INVALID_VALUE

ulTotalNumBytes 32-bit value

This is the total number of bytes that need to be retrieved to recreate the information recorded by the chip for the debug channel, which can then be stored in a binary file.

Direction: OUT Type: UINT32

Default: cOCT6100_INVALID_VALUE

ulMaxBytes 32-bit value

Maximum number of bytes that the user is ready to accept from the API. This represents the size of the buffer pointed by **pbyData**. This value must be modulo 1024.

Direction: IN Type: UINT32

Default: cOCT6100_INVALID_VALUE

ulValidNumBytes 1 - **ulMaxBytes**

This is the number of bytes that are valid in the **pbyData** array. This value ranges between 1 and **ulMaxBytes**.

Direction: OUT Type: UINT32

Default: cOCT6100_INVALID_VALUE

pbyData pointer to array

Byte pointer to a memory block where the recorded data will be copied to. The user must allocate this memory.

Direction: IN/OUT Type: PUINT8

Default: NULL

Sample Code

The sample code below illustrates how the debug functions should be used:

```
void Oct6100GetDumpDataExample()
{
    tOCT6100_DEBUG_GET_DATA             GetData;
    tOCT6100_DEBUG_SELECT_CHANNEL       SelectChannel;
    UINT32                              ulResult;
    int                                 iNumWritten;
    FILE *                              pDumpFile;

    Oct6100DebugSelectChannelDef( &SelectChannel );

    /* Channel handle (returned from a call to Oct6100ChannelOpen) */
    /* of the channel to be monitored (hot channel). */
    SelectChannel.ulChannelHndl = g_ulChannelHndl;

    /* Select the debug channel.  The chip firmware will start recording */
    /* information on this channel.  PCM data will be accumulated in the */
    /* chip's external memory for up to 2 minutes. */
    ulResult = Oct6100DebugSelectChannel( pApiInstance, &SelectChannel );
    if ( ulResult != cOCT6100_ERR_OK )
    {
        /* Error handling. */
        return;
    }

    /* Wait 16 seconds for data to be recorded on the current debug channel. */
    /* During this time, information is recorded on the debug channel. */
    /* The user could, for example, play a test case buffer on the debug channel's */
    /* inputs to record how the chip reacts.  When this is done, */
    /* the information can be dumped in a file and sent to Octasic for analysis. */
    Sleep( 16 * 1000 );

    /* Open binary file that will receive dump information to be sent to Octasic. */
    pDumpFile = fopen( "oct61xx_support.bin", "wb" );
    if ( pDumpFile == NULL )
    {
        /* Error handling. */
        return;
    }
    Oct6100DebugGetDataDef( &GetData );

    /* Select how much data should be recorded. */
    GetData.ulGetDataMode = cOCT6100_DEBUG_GET_DATA_MODE_16S_LITE;

    /* Other available choices are:  */
    /* cOCT6100_DEBUG_GET_DATA_MODE_120S */
```

```
    /* cOCT6100_DEBUG_GET_DATA_MODE_16S */
    /* cOCT6100_DEBUG_GET_DATA_MODE_120S_LITE */
    /* cOCT6100_DEBUG_GET_DATA_MODE_CORE_DUMP */

    /* Set number of bytes available in transfer buffer. */
    GetData.ulMaxBytes = 2048 * 50;

    /* Allocate memory for the transfer buffer. */
    GetData.pbyData = (PUINT8)malloc( GetData.ulMaxBytes );
    if ( GetData.pbyData == NULL )
    {
        /* Error handling. */
        fclose( pDumpFile );
        return;
    }

    /* Read all the dump information in a loop. */
    /* Transfer at most GetData.ulMaxBytes at a time. */
    do
    {
        /* Call the API function for retrieving the dump information. */
        ulResult = Oct6100DebugGetData( pApiInstance, &GetData );
        if ( ulResult != cOCT6100_ERR_OK )
        {
            /* Error handling. */
            break;
        }

        /* Write the data at the end of the dump file. */
        iNumWritten = fwrite( GetData.pbyData, 1, GetData.ulValidNumBytes, pDumpFile );
        if ( iNumWritten != (int)GetData.ulValidNumBytes )
        {
            /* Error handling. */
            break;
        }

        /* Repeat this until all the bytes have been read. */

    } while ( GetData.ulRemainingNumBytes != 0x0 );

    /* Free the transfer buffer. */
    free( GetData.pbyData );

    /* Close the dump file. */
    fclose( pDumpFile );
}
```

Once the data of a complete dump (cOCT6100_DEBUG_GET_DATA_CONTENT_COMPLETE) has been recorded to the file, the Oct6100 Remote Client application can be used to extract the PCM data files from the binary diagnostic file. Calling the executable with the binary diagnostic file as an argument does this. Refer to the Remote Client User guide (Octasic literature number **oct6100ug5000**) for more information.

6 User Supplied Functions Description

Several user-supplied functions are required for the API to be independent from the target system and OS. This includes functions for process serialization, access to the current time, and device access routines.

6.1 Serialization Functions

The API code needs the ability to serialize access to several internal structures. This serialization can be performed using a semaphore, mutex, or any other serialization method. A single serialization object is required by the API, the user supplies functions to create and destroy this object, as well as functions to seize and release it.

Please note that the API's code is written with the assumption that the underlying OS uses priority inheritance for threads using the API. That is, if Thread 1 is scheduled as a low priority thread and is currently using an exclusive resource that Thread 2, a thread with a higher priority, needs to go on with the execution, then Thread 1's priority will be increased to that of Thread 2 to avoid deadlock.

6.1.1 Oct6100UserCreateSerializeObject

This function creates a user-supplied serialization object. The serialization object can be a semaphore, mutex, or any other form of serialization. A handle that identifies the created object is returned. The returned handle is used in any subsequent call that affects the created serialization object.

Usage

```
#include "oct6100_apiud.h"

UINT32 Oct6100UserCreateSerializeObject(
            tPOCT6100_CREATE_SERIALIZE_OBJECT        f_pCreate );
```

Parameters

f_pCreate	Pointer to a tOCT6100_CREATE_SERIALIZE_OBJECT structure. The structure's elements are defined below.

6.1.1.1 tOCT6100_CREATE_SERIALIZE_OBJECT Structure

ulSerialObjHndl 32-bit value

Handle returned by this routine to identify the created serialization object in future calls that affects it.

Direction: OUT Type: tOCT6100_USER_SERIAL_OBJECT

Default: N/A

6.1.2 Oct6100UserDestroySerializeObject

Destroys the user serialization object created using **Oct6100UserCreateSerializationObject** and identified by the provided handle.

Usage

#include "oct6100_apiud.h"

UINT32 Oct6100UserDestroySerializeObject (
tPOCT6100_DESTROY_SERIALIZE_OBJECT f_pDestroy);

Parameters

f_pDestroy	Pointer to a tOCT6100_DESTROY_SERIALIZE_OBJECT structure. The structure's elements are defined below.

6.1.2.1 tOCT6100_DESTROY_SERIALIZE_OBJECT Structure

ulSerialObjHndl 32-bit value

Pointer to the handle returned from the call to the **Oct6100UserCreateSerializationObject** function that created the object.

Direction: IN Type: tOCT6100_USER_SERIAL_OBJECT

Default: N/A

6.1.3 Oct6100UserSeizeSerializeObject

Seizes the serialization object indicated by the provided handle. The routine attempts to seize the semaphore for the specified amount of time before returning without success.

Usage

#include "oct6100_apiud.h"

UINT32 Oct6100UserSeizeSerializeObject(
tPOCT6100_SEIZE_SERIALIZE_OBJECT f_pSeize);

Parameters

f_pSeize — Pointer to a tOCT6100_SEIZE_SERIALIZE_OBJECT structure. The structure's elements are defined below.

6.1.3.1 tOCT6100_SEIZE_SERIALIZE_OBJECT Structure

ulSerialObjHndl — 32-bit value

Pointer to the handle returned from the call to the **Oct6100UserCreateSerializationObject** function that created the object.

Direction: IN — Type: tOCT6100_USER_SERIAL_OBJECT
Default: N/A

ulTryTimeMs — 32-bit value, cOCT6100_WAIT_INFINITELY

The period, in ms, during which the routine attempts to seize the serialization object before it returns without success. If equal to 0, the function attempts to seize the semaphore only once.

If set to cOCT6100_WAIT_INFINITELY, the function does not return until the serialization object is seized.

Direction: IN — Type: UINT32
Default: N/A

6.1.4 Oct6100UserReleaseSerializeObject

Releases the serialization object indicated by the provided handle and seized using the **Oct6100UserSeizeSerializationObject** function.

Usage

#include "oct6100_apiud.h"

UINT32 Oct6100UserReleaseSerializeObject(
tPOCT6100_RELEASE_SERIALIZE_OBJECT f_pRelease);

Parameters

f_pRelease — Pointer to a tOCT6100_RELEASE_SERIALIZE_OBJECT structure. The structure's elements are defined below.

6.1.4.1 tOCT6100_RELEASE_SERIALIZE_OBJECT Structure

ulSerialObjHndl 32-bit value

Pointer to the handle returned from the call to the **Oct6100UserCreateSerializationObject** function that created the object.

Direction: IN Type: tOCT6100_USER_SERIAL_OBJECT

Default: N/A

6.2 Write Functions

6.2.1 Oct6100UserDriverWriteApi, Oct6100UserDriverWriteOs

Performs a single word write to the chip. Any error returned by the function is considered a fatal error. Two versions of the function are needed because the function may be accessed from two different software layers. Refer to the **System Architecture** description provided in the **Overview** section. Thus, each function must have a different name, but the functionality remains identical.

Usage

```
#include "oct6100_apiud.h"

UINT32 Oct6100UserWriteApi(
        tPOCT6100_WRITE_PARMS        f_pWriteParms );

UINT32 Oct6100UserWriteOs(
        tPOCT6100_WRITE_PARMS        f_pWriteParms );
```

Parameters

f_pWriteParms — Pointer to a tOCT6100_WRITE_PARMS structure. The structure's elements are defined below.

6.2.1.1 tOCT6100_WRITE_PARMS Structure

pProcessContext pointer

This parameter is used only if **fMultiProcessSystem** is set to TRUE in the tOCT6100_OPEN_CHIP structure. Pointer to structure provided by user during creation of local API instance.

Direction: IN Type: PVOID
Default: N/A

ulUserChipId identifier

The chip identifier parameter provided to the Oct6100ChipOpen function. (see **System Architecture**).

Direction: IN Type: UINT32
Default: N/A

ulWriteAddress 0 – 0x0FFFFFFE

Start address of the word access. This address is in bytes but be on a word boundary.

Direction: IN Type: UINT32
Default: N/A

usWriteData 16 bit field

This is the word to be written by this function call.

Direction: IN Type: UINT16
Default: N/A

6.2.2 Oct6100UserDriverWriteSmearApi, Oct6100UserDriverWriteSmearOs

Performs a write of the same data word to multiple addresses of the chip. Any error returned by this function is considered a fatal error. One or two versions of the function are needed because the function may be accessed from one or two different software layers, depending on the user system architecture. See the **System Architecture** description in the **Overview**. Thus, each function must have a different name, but the functionality remains identical.

Usage

```
#include "oct6100_apiud.h"

UINT32 Oct6100UserWriteSmearApi(
            tPOCT6100_WRITE_SMEAR_PARMS        f_pWriteSmearParms );

UINT32 Oct6100UserWriteSmearOs(
            tPOCT6100_WRITE_SMEAR_PARMS        f_pWriteSmearParms );
```

Parameters

f_pWriteSmearParms — Pointer to a tOCT6100_WRITE_SMEAR_PARMS structure. The structure's elements are defined below.

6.2.2.1 tOCT6100_WRITE_SMEAR_PARMS Structure

pProcessContext pointer

This parameter is used only if **fMultiProcessSystem** is set to TRUE in the tOCT6100_OPEN_CHIP structure. Pointer to structure provided by user during creation of local API instance.

Direction: IN Type: PVOID
Default: N/A

ulUserChipId identifier

The chip identifier parameter provided to the Oct6100ChipOpen function. (see **System Architecture**).

Direction: IN Type: UINT32
Default: N/A

ulWriteAddress 0 – 0x0FFFFFFE

Start address of the writes. This is a byte address that points to words and must be even. This is the address of the first location. For each subsequent word, the address is incremented by two.

Direction: IN Type: UINT32
Default: N/A

usWriteData 16 bit field

The word to be written.

Direction: IN Type: UINT16
Default: N/A

ulWriteLength 32-bit value

The number of accesses that must be performed.

Direction: IN Type: UINT32

Default: N/A

6.2.3 Oct6100UserDriverWriteBurstApi, Oct6100UserDriverWriteBurstOs

Writes an array of data words to consecutive addresses of the chip. Any error returned by the function is considered a fatal error. Two versions of the function are needed because the function may be accessed from two different software layers. See the **System Architecture** description in the **Overview**. Thus, each function must have a different name, but the functionality remains identical.

Usage

```
#include "oct6100_apiud.h"

UINT32 Oct6100UserWriteBurstApi(
            tPOCT6100_WRITE_BURST_PARMS        f_pWriteBurstParms );

UINT32 Oct6100UserWriteBurstOs(
            tPOCT6100_WRITE_BURST_PARMS        f_pWriteBurstParms );
```

Parameters

f_pWriteBurstParms — Pointer to a tOCT6100_WRITE_BURST_PARMS structure. The structure's elements are defined below.

6.2.3.1 tOCT6100_WRITE_BURST_PARMS Structure

pProcessContext pointer

This parameter is used only if **fMultiProcessSystem** is set to TRUE in the tOCT6100_OPEN_CHIP structure. Pointer to structure provided by user during creation of local API instance.

Direction: IN Type: PVOID
Default: N/A

ulUserChipId identifier

The chip identifier parameter provided to the Oct6100ChipOpen function. (see **System Architecture**).

Direction: IN Type: UINT32
Default: N/A

ulWriteAddress 0 – 0x0FFFFFFE

Start address of the writes. This is a byte address that points to words and must be even. This is the address of the first location to write to. For each subsequent word the address is incremented by two.

Direction: IN Type: UINT32
Default: N/A

pusWriteData

Array of words to be written starting at **ulWriteAddress**.

Direction: IN Type: PUINT16
Default: N/A

ulWriteLength 32-bit value

The number of accesses that must be performed.

Direction: IN Type: UINT32

Default: N/A

6.3 Read Functions

6.3.1 Oct6100UserDriverReadApi, Oct6100UserDriverReadOs

Reads a single word from the chip. Any error returned by the function is considered a fatal error. Two versions of the function are needed because the function may be accessed from two different software layers. See the **System Architecture** description in the **Overview**. Thus, each function must have a different name, but the functionality remains identical.

Usage

```
#include "oct6100_apiud.h"

UINT32 Oct6100UserReadApi(
        tPOCT6100_READ_PARMS        f_pReadParms );

UINT32 Oct6100UserReadOs(
        tPOCT6100_READ_PARMS        f_pReadParms );
```

Parameters

f_pReadParms — Pointer to a tOCT6100_READ_PARMS structure. The structure's elements are defined below.

6.3.1.1 tOCT6100_READ_PARMS Structure

pProcessContext pointer

This parameter is used only if **fMultiProcessSystem** is set to TRUE in the tOCT6100_OPEN_CHIP structure. Pointer to structure provided by user during creation of local API instance.

Direction: IN Type: PVOID
Default: N/A

ulUserChipId identifier

The chip identifier parameter provided to the Oct6100ChipOpen function. (see **System Architecture**).

Direction: IN Type: UINT32
Default: N/A

ulReadAddress 0 – 0x0FFFFFFE

This is the address of the word to be read. This is a byte address that must be on a word boundary.

Direction: IN Type: UINT32
Default: N/A

pusReadData

Pointer to a single UINT16 to receive the data.

Direction: IN/OUT Type: PUINT16
Default: N/A

6.3.2 Oct6100UserDriverReadBurstApi, Oct6100UserDriverReadBurstOs

Performs a burst of reads to the chip. Any error returned by this function is considered a fatal error. Two versions of the function are needed because the function may be accessed from two different software layers. See the **System Architecture** description in the **Overview**. Thus, each function must have a different name, but the functionality remains identical.

Usage

```
#include "oct6100_apiud.h"

UINT32 Oct6100UserReadBurstApi(
            tPOCT6100_READ_BURST_PARMS          f_pReadBurstParms );

UINT32 Oct6100UserReadBurstOs(
            tPOCT6100_READ_BURST_PARMS          f_pReadBurstParms );
```

Parameters

f_pReadBurstParms — Pointer to a tOCT6100_READ_BURST_PARMS structure. The structure's elements are defined below.

6.3.2.1 tOCT6100_READ_BURST_PARMS Structure

pProcessContext pointer

This parameter is used only if **fMultiProcessSystem** is set to TRUE in the tOCT6100_OPEN_CHIP structure. Pointer to structure provided by user during creation of local API instance.

Direction: IN Type: PVOID
Default: N/A

ulUserChipId identifier

The chip identifier parameter provided to the Oct6100ChipOpen function. (see **System Architecture**).

Direction: IN Type: UINT32
Default: N/A

ulReadAddress 0 – 0x0FFFFFFE

Start address of the burst. This is a byte address that must be on a word boundary. This is the address of the first word in the burst. For each subsequent word, the address is incremented by two.

Direction: IN Type: UINT32
Default: N/A

pusReadData

Pointer to an array of UINT16s used to receive the data read. Each element is one word.

Direction: IN/OUT Type: PUINT16
Default: N/A

ulReadLength 32-bit value

Length of the **pulReadData** (burst length in words).

Direction: IN Type: UINT32

Default: N/A

6.4 Time Functions

The API requires wall-clock time, in microseconds (µs) for resource and event management. There are resources that require a minimum invalid time before reuse to guarantee proper operation.

6.4.1 Oct6100UserGetTime

Gets the current value of the user supplied wall-clock time. The time is specified in µs. It is important that this timer never wrap. Of course, if the counter is initialized to 0 before the API calls this function then the counter will only wrap in several thousand years, thus insuring that no wrapping occurs. The API code is written with the assumption that the counter will never wrap.

Usage

```
#include "oct6100_apiud.h"

UINT32 Oct6100UserGetTime(
            tPOCT6100_GET_TIME      f_pGetTime );
```

Return Values

COCT6100_GET_TIME_FAILED_*X*	the values 0xFFFF0000 - 0xFFFF000F are reserved for time routine return values. The *x* is the last hex digit of the returned value. This return value is passed by the API function to the original calling user routine. Any error returned in this range by this function is considered a fatal error by the API.

Parameters

f_pGetTime	pointer to a tOCT6100_GET_TIME structure. The definitions of the structure's elements are listed below.

6.4.1.1 Structure tOCT6100_GET_TIME

pProcessContext pointer

This parameter is used only if **fMultiProcessSystem** is set to TRUE in the tOCT6100_OPEN_CHIP structure. Pointer to structure provided by user during creation of local API instance.

Direction: IN Type: PVOID
Default: N/A

ulWallTimeUs[2] UINT32

The returned wall-time in µs. Element 1 of the array contains the MSB bits.

Direction: OUT Type: UINT32[2]
Default: N/A

6.5 Memory Functions

The API requires some simple memory management functions from the user. The required functions are **Oct6100UserMemSet** and **Oct6100UserMemCopy**.

6.5.1 Oct6100UserMemSet

This function is used to set a memory space to a specified value.

Usage

```
#include "oct6100_apiud.h"

UINT32 Oct6100UserMemSet(
            PVOID           f_pAddress,
            UINT32          f_ulPattern,
            UINT32          f_ulLength );
```

Parameters

f_pAddress	Pointer to the memory segment where f_ulPattern will be written.
f_ulPattern	Byte pattern written into memory.
f_ulLength	Number of bytes after f_pAddress that must be written to f_ulPattern.

6.5.2 Oct6100UserMemCopy

This function is used to copy data from one memory location to another.

Usage

```
#include "oct6100_apiud.h"

UINT32 Oct6100UserMemCopy(
            PVOID           f_pDestination,
            PVOID           f_pSource,
            UINT32          f_ulLength );
```

Parameters

f_pDestination	Pointer to the memory segment where the information needs to be copied.
f_pSource	Pointer to the memory segment where the information to be copied is located.
f_ulLength	Number of bytes to be copied.

7 Echo Operation Mode

The **ulEchoOperationMode** parameter of the channel configuration structure determines the state of the echo canceller. The **fEnableNlp** parameter of the channel configuration structure determines the state of the Non-Linear processor (NLP). Certain features are not available in certain modes.

The table below indicates which features are available in each of the echo operation modes and NLP states. An "X" in a box indicates that the feature IS available.

Note that the table below lists all features that the OCT61xx can support. Not all features are available for all OCT61xx devices. Please refer to the OCT6100 Hardware specification for the feature list for your specific part.

MODES	**ECHO OPERATION MODE**						**NLP ENABLE**	
FEATURES	**POWER DOWN**	**HT FREEZE**	**HT RESET**	**NO ECHO NLP ON**	**SPEECH RECOG. NLP OFF**	**NORMAL**	**FALSE**	**TRUE**
ADPCM	X	X	X	X	X	X	X	X
Broadcast TSST	X	X	X	X	X	X	X	X
Conferencing	X	X	X	X	X	X	X	X
Law Conversion	X	X	X	X	X	X	X	X
Phasing	X	X	X	X	X	X	X	X
TSI Connection	X	X	X	X	X	X	X	X
Tone Detection		X	X	X	X	X	X	X
DC Removal		X	X	X	X	X	X	X
MLC		X	X	X	X	X	X	X
Silence Supp.		X	X	X	X	X		X
ANR		X	X	X	X	X		X
CNR		X	X	X	X	X		X
Tone Removal		X	X	X	X	X		X
Dominant speaker		X	X	X	X	X		X
Buffer Playout			X	X	X	X		X
ALC				X	X	X		X
HLC				X	X	X		X
Sin Voice Detection				X	X	X		X
Tone Disabler		X	X		X	X	X	X
Default ERL		X	X			X		X
NLP Behavior A		X	X			X		X
Echo Cancellation					X	X	X	X
AEC					X	X	X	X
Comfort Noise						X		X
Tail Displacement					X	X	X	X
NLP Behavior B						X		X
Idle code detection						X		X
Octasic Music Protection						X		X

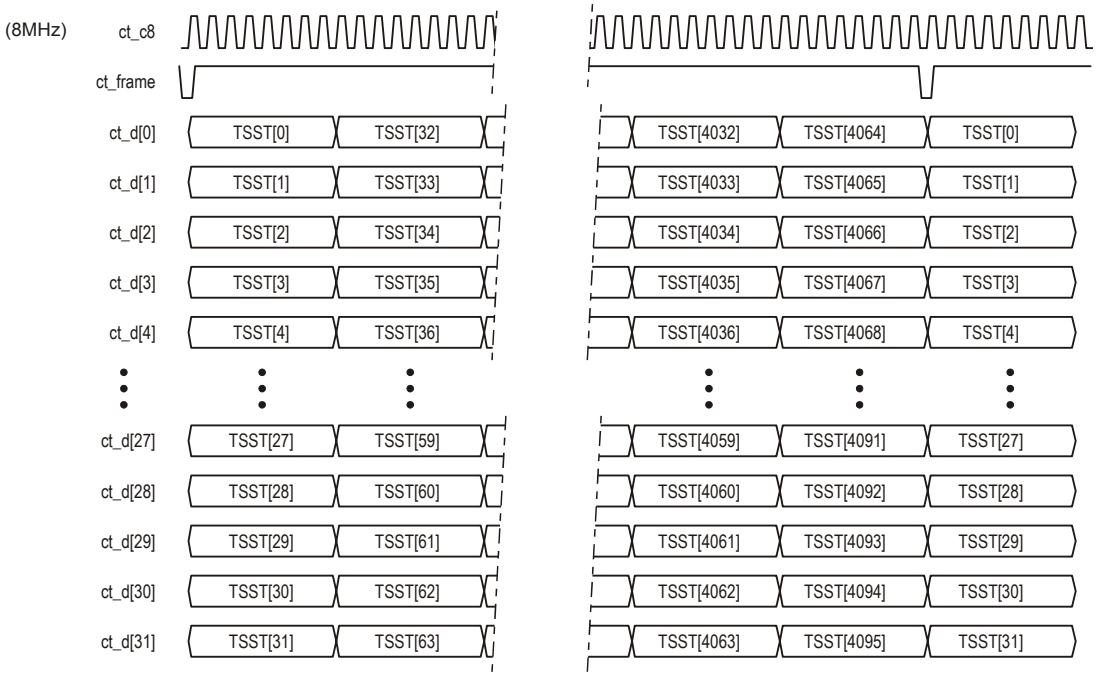
8 API access count per function

This section provides information about the number of writes and reads perform by API functions.

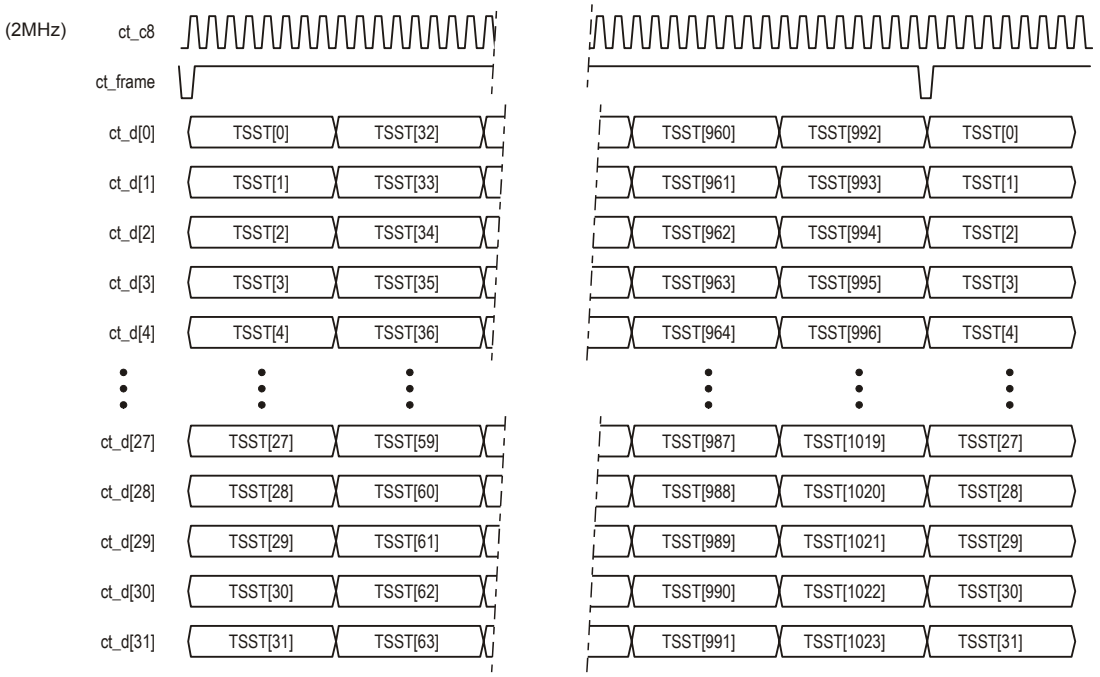
Function Name	Characteristics		Num Write Word	Num Read Word
Oct6100ChipOpen	image of 238997 bytes		136249	16152
Oct6100ChipGetStats			0	0
Oct6100ChipGetImageInfo			0	0
Oct6100ChannelOpen	with ADPCM		84	20
Oct6100ChannelOpen	without ADPCM		76	20
Oct6100ChannelModify	Modify Op mode only		66	0
Oct6100ChannelModify	Modify VQE only		38	0
Oct6100ChannelModify	Modify Codec only		46	0
Oct6100ChannelModify	Modify TDM only		46	0
Oct6100ChannelModify	Modify All		66	0
Oct6100ChannelGetStats			0	11
Oct6100ChannelClose			37	2
Oct6100InterruptServiceRoutine	No Tone events		5	15
Oct6100ToneDetectionEnable			1	1
Oct6100EventGetTone	No Tone Events		1	2
Oct6100EventGetTone	1 Tone Event		1	34
Oct6100ToneDetectionDisable			3	3
Oct6100ConfBridgeOpen			0	0
Oct6100ConfBridgeChanAdd			25	0
Oct6100ConfBridgeChanRemove			10	0
Oct6100ConfBridgeGetStats			0	0
Oct6100ConfBridgeClose			0	0
Oct6100BufferPlayoutLoad	16384-byte buffer		8192	0
Oct6100BufferPlayoutAdd			8	2
Oct6100BufferPlayoutStart			4	0
Oct6100BufferPlayoutStop			8	0
Oct6100BufferPlayoutUnload			0	0

9 TSST to Timeslot Mapping

128 timeslots per frame



32 timeslots per frame



10 TSST Formats

The OCT6100 supports auto-detection of the compression format in the input direction and notifies an external SAR device on the current compression rate and silence indications in output direction.

When the formats allow changes between PCM, G.726/G.727 and silence, additional encodings and potentially TSSTs are required to carry complete information for a single channel on the H.100 bus.

10.1 Input TSST Formats

10.1.1 One TSST Format

1 TSST Format Decompression Formats

Format of Fixed Rate Samples

b7	b6	b5	b4	b3	b2	b1	b0	
PCM 64kbps								
x	x	x	ADPCM 40kbps					
x	x	x	x	32kbps				
x	x	x	x	x	24kbps			
x	x	x	x	x	x	16kbps		

Format of Auto Rate Detected Samples

b7	b6	b5	b4	b3	b2	b1	b0	
PCM[7]	1	PCM[5:0]						
R	0	1	ADPCM 40kbps					
R	0	0	1	32kbps				
R	0	0	0	1	24kbps			
R	0	0	0	0	1	16kbps		

Note: This format is intended for comfort noise samples from the SAR device. Since comfort noise should be rather low in energy, bit 6 is always 1 in u-law and A-law. (u-law max value of 471 / 8159. A-law max value of 252 / 4096).

R - Reset ADPCM Codec, when set to '1' the codec is reset before processing the sample. When the format changes from PCM to any ADPCM format, a reset is performed on the first ADPCM sample whether indicated or not.

10.1.2 Two TSST Format

2 TSST Format Decompression Formats

Associated TSST (Configured TSST - 1)								Configured TSST (odd)							
b7	b6	b5	b4	b3	b2	b1	b0	b7	b6	b5	b4	b3	b2	b1	b0
1	R	Reserved						PCM Sample							
0	0	Reserved						R	0	1	ADPCM 40kbps				
0	0	Reserved						R	0	0	1	32kbps			
0	0	Reserved						R	0	0	0	1	24kbps		
0	0	Reserved						R	0	0	0	0	1	16kbps	

R - Reset ADPCM Codec, when set to '1' the codec is reset before processing the sample. When the format changes from PCM to any ADPCM format, a reset is performed on the first ADPCM sample whether indicated or not.

10.2 Output TSST Formats

10.2.1 One TSST Format

1 TSST Format Compression Formats

Silence Suppression Indication

b7	b6	b5	b4	b3	b2	b1	b0
0	0	0	0	0	0	0	0

Note: This indication will be presented in the frame of the sample that would end a packet as determined by the phasing TSST and the phase of the channel. This indicates that the OCT6100 voice activity detector has determined that this packet should be suppressed. In One TSST mode, if the channel has been configured for PCM, then silence suppression will not operate properly.

Format of Samples

b7	b6	b5	b4	b3	b2	b1	b0
PCM 64kbps							

b7	b6	b5	b4	b3	b2	b1	b0
0	0	1	ADPCM 40kbps				

b7	b6	b5	b4	b3	b2	b1	b0
0	0	0	1	32kbps			

b7	b6	b5	b4	b3	b2	b1	b0
0	0	0	0	1	24kbps		

b7	b6	b5	b4	b3	b2	b1	b0
0	0	0	0	0	1	16kbps	

10.2.2 Two TSST Format

2 TSST Format Compression Formats

Associated TSST (Configured TSST - 1)								Configured TSST (odd)							
b7	b6	b5	b4	b3	b2	b1	b0	b7	b6	b5	b4	b3	b2	b1	b0
1	TX PCM Unsigned Mag							PCM Sample							
0	TX PCM Unsigned Mag							0	0	1	ADPCM 40kbps				
0	TX PCM Unsigned Mag							0	0	0	1	32kbps			
0	TX PCM Unsigned Mag							0	0	0	0	1	24kbps		
0	TX PCM Unsigned Mag							0	0	0	0	0	1	16kbps	

Silence Suppression Indication

b7	b6	b5	b4	b3	b2	b1	b0	b7	b6	b5	b4	b3	b2	b1	b0
0	TX PCM Unsigned Mag							0	0	0	0	0	0	0	0

Note: This indication will be presented in the frame of the sample that would end a packet as determined by the phasing TSST and the phase of the channel. This indicates that the OCT6100 voice activity detector has determined that this packet should be suppressed.

11 Revision History

Version	Date	Changes
4.0	December 2009	• Added dynamic enabling/disabling of channel recording in monitoring section.
3.9	September 2009	• Added a new core dump data mode in the monitoring section.
3.8	December 2008	• Updated Conference bridge section • Updated Buffer Playout section
3.7	September 2008	• Removed cOCT6100_MIXING_0_DB from ulMixingMode • Added multi-process system description in API architecture section. • Added sample code for multi-process applications. • Modified description of pulDurationMs parameter in Oct6100CallerIdTransmit function. • Added Adaptive Filter dynamic BIST feature.
3.6	June 2007	• Tone Event Delay Modification. • Added fEnable2100StopEvent. • Removed ALC tone reset and ALC noise bleed timeout features.
3.5	December 2006	• Added fRoutNoiseReductionLevel and lRoutNoiseReductionLevelGainDb.
3.4	November 2006	• Added Oct6100ApiGetCapacityPins function.
3.3	October 2006	• Added new shorter production BIST mode.
3.2	April 2006	• New "content" parameter to the debug recording functions.
3.1	March 2006	• New OCT6170 product family.
3.0	February 2006	• New RIN and SOUT ALC reset parameters upon detection of continuous tones. • Added ALC noise bleed out time parameter. • New RIN port DTMF tone removal feature using bi-directional channel linking.
2.9	November 2005	• Added source code example for debug recording functions.
2.8	September 2005	• Added idle code detection parameter.
2.7	July 2005	• Added double talk behavior parameter. • Added SOUT noise bleaching parameter. • Changed the default non-linearity behavior A to 1.
2.6	April 2005	• Added tone profile number parameter description to the chip image info.
2.5	February 2005	• Default ERL new supported values: -9 and -12.
2.4	January 2005	• New speech recognition echo operation mode. • New per channel tail length parameter. • New tone disabler VQE active delay. • New RIN port DTMF tone removal feature. • No more array upper limits on the event functions. • New per channel acoustic echo tail length parameter.

		• New production BIST function. • New music protection flag in channel functions. • New debug recording functions.
2.3	November 2004	• New adaptive noise reduction parameters. • New modify channel flag to apply the configuration to all opened channels. • New modify channel flags to stop buffer playout, disable tone detection, remove a participant from a bridge and close all broadcast TSSTs.
2.2	September 2004	• New caller ID stop events.
2.1	August 2004	• New listener enhancement parameters. • New ROUT noise reduction feature.
2.0	July 2004	• New modify of channel number of TSSTs. • New channel mute and un-mute functions.
1.9	June 2004	• New tap feature in simple conferencing.
1.8	June 2004	• New image info flag for the dominant speaker feature. • New channel applied gain statistics.
1.7	April 2004	• New buffer playout parameter that controls the gain. • New buffer playout 0 dB mixing mode. • New conference bridge function for changing the participant mask. • New Automatic Level Control parameters. • New High Level Compensation parameters. • New echo operation mode that activates the voice quality features without echo cancellation. • New parameters to the buffer playout stop function.
1.6	March 2004	• Caller ID specification. • Acoustic echo cancellation not using non-linearity behavior B and tail displacement parameters. • New acoustic default ERL parameter. • Image info structure modifications. • New conferencing noise reduction feature. • Changed the minimum playout buffer size to 64.

For more information on this or other products visit our web site: http://www.octasic.com

Or contact us at:

Tel: +1 (514) 282-1361

Fax: +1 (514) 282-7672

Email: support@octasic.com

Address:

OCTASIC Inc.
4101 Molson St, Suite 300
Montreal, Quebec
H1Y 3L1, Canada