

**EECS 123:**  
**Introduction to  
Real-Time Distributed Programming**  
**Lecture : RMMC2SvM**

Feb-07



## RMMC2SvM

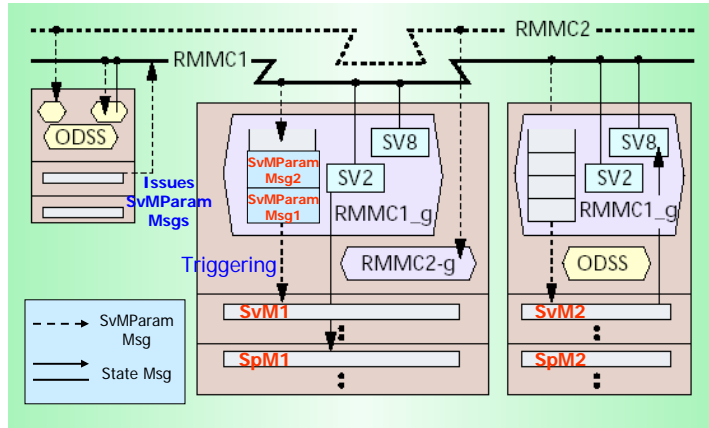
---

RMMC2SvM := RMMC event msg triggering SvM(s) +  
RMMC state msg

Feb-07



# RMMC2SvM



- As in the case of standard RMMC, access gates for 2 RMMC2SvMs (RMMC1 and RMMC2) can be declared as **data members** (special types of ODSSs) of each of the 3 TMOs during the design time.
- Each gate is treated as a subscriber to the associated RMMC2SvM channel. One TMO can have multiple gates for the same RMMC2SvM channel.

Feb-07

DREAM Lab



## SvM Parameter Messages & State Messages

- RMMC2SvM scheme supports (1) **state messages** based on **distributed replicated memory semantics** and (2) **SvM parameter messages** but no event messages.
- **State messages** in RMMC2SvM are exactly the same as those in standard RMMC.
- **SvM Parameter Message**
  - Registration of an RMMC2SvM-Gate involves registration of a **SvM(s)** to be triggered by any SvM parameter message carried by the RMMC2SvM.
  - SvM parameter messages are sent to all subscriber RMMC2SvM-Gates except the sender RMMC2SvM-Gate.
  - When an SvM parameter message reaches the TMO SM node hosting a subscriber RMMC2SvM-Gate, the message is kept inside TMO SM without being released to the gate until its **OfficialReleaseTime** arrives.
  - When the **OfficialReleaseTime** of an SvM parameter msg arrives, the msg in each receiving RMMC2SvM-Gate hosted in a TMO is consumed by the engine with an effect that an execution of the **SvM(s) in the TMO associated with the gate is triggered**.

Feb-07

UCI  
DREAM Lab



## Official Release Time (ORT)

---

- The **official release time** (ORT) is the time at which the SvM parameter / state message should become **accessible** through subscriber RMMC-Gates to the consumer methods.
- Registered SvMs are triggered by **SvM parameter messages in the order of their ORTs**.
- Consumer methods in each TMO with a subscriber RMMC-Gate can read only **the most recent officially released state messages** kept in the state msg memories associated with the gate.
- If **ORT is not given by the sender**, messages are released to receiving subscribers **ASAP**.

Feb-07

UCI  
DREAM Lab



## Official Release Time (ORT)

---

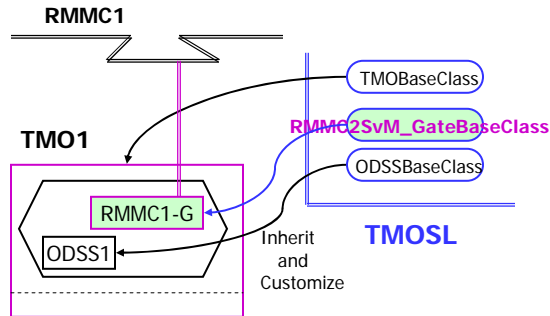
- **Toshiba has a patent on the idea of using ORT !**
  - Invented in 1996 and US Patent was granted in April 2001 but we learned the invention only in February 2005 although we have been using it since mid-1999.
  - As will be discussed later, ORT can also be emulated at the application level .

Feb-07

UCI  
DREAM Lab



## Creation of an RMMC2SvM Access Gate & an Enclosing TMO



Feb-07

UCI  
DREAM Lab



## Differences between RMMC2SvM and OnewaySR

### 1. Number of SvMs to be Triggered

- In the case of OnewaySR, **only one SvM can be triggered by one OnewaySR call.**
- In the case of RMMC-based triggering of SvMs, **multiple SvMs may be bound to the same RMMC2SvM channel,** although use of such feature may be rare, and all SvMs bound to the channel can be triggered after a given official release time.  
Also, one SvM may be bound to more than one RMMC2SvM.

### 2. Message Size – minor issue

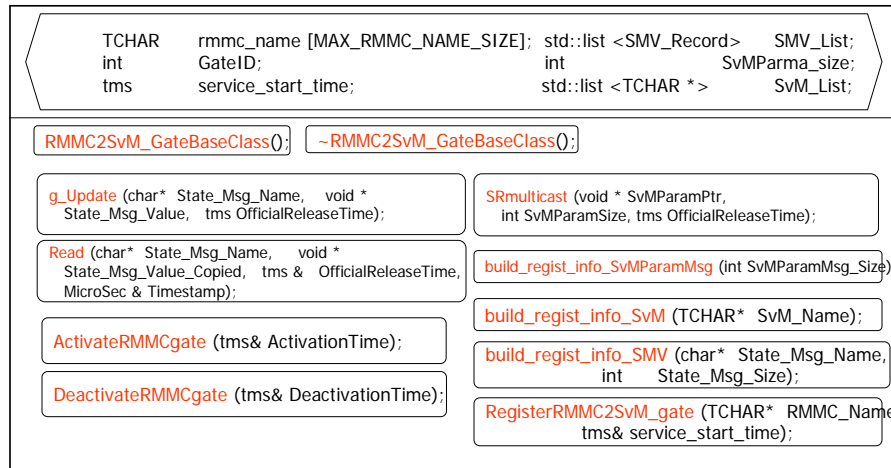
- The size of an OnewaySR message needs not be fixed until the SR is issued, although it is normally fixed at the time of defining the SvM.
- The size of an SvM Parameter Message becomes **fixed when the carrier RMMC2SvM is registered into TMOSM.**

Feb-07

UCI  
DREAM Lab



## RMMC2SvM\_GateBaseClass



Feb-07

UCI  
DREAM Lab



## API's

- Common APIs for establishing an RMMC2SvM

```
// Initialize the data members of RMMC2SvM_GateBaseClass
- RMMC2SvM_GateBaseClass::RMMC2SvM_GateBaseClass ()

// Remove the RMMC2SvM gate from the list of gates which connect to
// the RMMC2SvM
- RMMC2SvM_GateBaseClass::~~RMMC2SvM_GateBaseClass ()

/* Prepare a copy of the subject State Message variable in a form for
registration inside a local (node) instantiation of TMOSM.

The first parameter, which is a character string, will be used as the
globally recognized name of the subject SM variable. */
- void RMMC2SvM_GateBaseClass::build_regist_info_SMV (
TCHAR * SM_name, int SM_size);
```

Feb-07

UCI  
DREAM Lab



## API's

- Common APIs for establishing an RMMC2SvM

```
// Prepare the size information of the SvM Parameter Message
// in a form for Registering within a local (node) instantiation
// of TMOSM
```

  - **void RMMC2SvM\_GateBaseClass::build\_regist\_info\_SvMParamMsg** (int SvMParamMsg\_size)

```
// Register SvM into RMMC2SvM channel
```
  - **void RMMC2SvM\_GateBaseClass::build\_regist\_info\_SvM** (TCHAR\* SvM\_Name)

```
// Create one support record inside TMOSM
// for a gate to the specified RMMC2SvM.
```
  - **BOOL RMMC2SvM\_GateBaseClass::RegisterRMMC2SvM\_gate** (TCHAR\* RMMC\_Name, tms & service\_start\_time)

Feb-07

UCI  
DREAM Lab



## API's (cont.)

- State Message API's

```
/* Perform a global update of the State Message variable.
I.E., Update all distributed replicas of the SM variable.
Note that an SM variable is referenced by a character string.
All TMOSM nodes hosting subscribers will update their copies of the SM
variables.

The "OfficialReleaseTime" indicates when the new value provided by
"State_Msg_Value_To_Be_Stored" should be released to each subscriber.
If "OfficialReleaseTime" is not provided, this state message will be released
to subscribers ASAP.

Returns SUCCESS or FAIL (when failure signals have been raised by TMOSM)
*/
```

  - int **RMMC2SvM\_GateBaseClass::g\_Update** (char\* StateMsg\_Name,  
void \* StateMsg\_Value\_To\_Be\_Stored,  
tms OfficialReleaseTime)

Feb-07

UCI  
DREAM Lab



## API's (cont.)

```
/* Read the local copy (kept in the local TMOSM supporting the subject  
RMMC-Gate) of the State Message variable.
```

```
After completion of this function, the result parameter "OfficialReleaseTime"  
contains the ORT attached to the value of the StateMsg variable which will  
be copied into the result parameter "State_Msg_Value_Copied".  
In case "OfficialReleaseTime" is 0, this state message is delivered to  
subscribers ASAP.
```

```
The result parameter "Timestamp" contains the "send-timestamp" created  
when this state message was sent out from the updating subscriber.
```

```
Note that this is a non-blocking function call.
```

```
Returns SUCCESS or FAIL (when failure signals have been raised by  
TMOSM). */
```

```
- int RMMC2SvM_GateBaseClass::Read (char* StateMsg_Name,  
void * StateMsg_Value_Copied,  
tms & OfficialReleaseTime,  
MicroSec & Timestamp)
```

Feb-07

UCI  
DREAM Lab



## API's (cont.)

```
/* Send out an SvM Parameter Message through an RMMC2SvM gate.  
The SvM Parameter Message will be received by all the subscriber  
RMMC2SvM-Gates except to the sending RMMC2SvM-Gate.
```

```
The relevant SvMs in the TMOs containing the receiving RMMC2SvM-Gates  
are triggered when the "OfficialReleaseTime" arrives.
```

```
If "OfficialReleaseTime" is not provided, this SvM parameter message will be  
released to all subscribers ASAP. */
```

```
int RMMC2SvM_GateBaseClass::SRmulticast (  
void * SvMParamPtr,  
int SvMParamSize,  
tms OfficialReleaseTime)
```

Feb-07

UCI  
DREAM Lab



## API's (cont.)

/\*This API will be used to disconnect the RMMC-Gate from the RMMC.  
Once an RMMC-Gate is deactivated, methods in the owner TMO can still access the RMMC-Gate but the gate does not provide a connection to the RMMC and thus can neither receive and announce Event messages nor read and g\_update State messages communicated through the RMMC.

Until Deactivation\_Time is reached, the RMMC gate remains connected to the RMMC. For receiving Event messages, those messages of which official release times (ORTs) are later than Deactivation time cannot be received. Announcement of Event messages can succeed if the messages pass through the RMMC-Gate before Deactivation\_Time.

For reading State messages, those messages of which ORTs are later than Deactivation\_Time cannot be read. g\_Update of State messages can succeed if the messages pass through the RMMC-Gate before Deactivation\_Time. \*/

```
- int RMMC2SvM_GateBaseClass::DeactivateRMMCgate (  
    tms & Deactivation_Time)
```

Feb-07

UCI  
DREAM Lab



## API's (cont.)

/\* This API will be used to connect the RMMC-Gate to the RMMC from which the gate was disconnected earlier.

Methods in the owner TMO can receive through the RMMC-Gate those Event messages of which Official Release Times (ORTs) are after Reactivation\_Time.

Announcement of Event messages can succeed if Announce () is called after Reactivation\_Time.

Similarly, the TMO methods can read State messages communicated through the RMMC and tagged ORTs which are after Reactivation\_Time.

g\_Update of State messages can succeed if g\_Update () is called after Reactivation\_Time. \*/

```
- int RMMC2SvM_GateBaseClass::ActivateRMMCgate (  
    tms & Reactivation_Time)
```

Feb-07

UCI  
DREAM Lab



## API's (cont.)

---

/\* A "send-timestamp" is created when an SvM parameter message is sent out through an RMMC2SvM as a result of an execution of SRmulticast ().

This timestamp can be retrieved on each server side by execution of the subject API GetSRTimestamp () inside the function body of a triggered SvM.

\*/

MicroSec **CTMOBase::GetSRTimestamp ()**

Feb-07

UCI  
DREAM Lab



## Example 1: Use of an RMMC2SvM

---

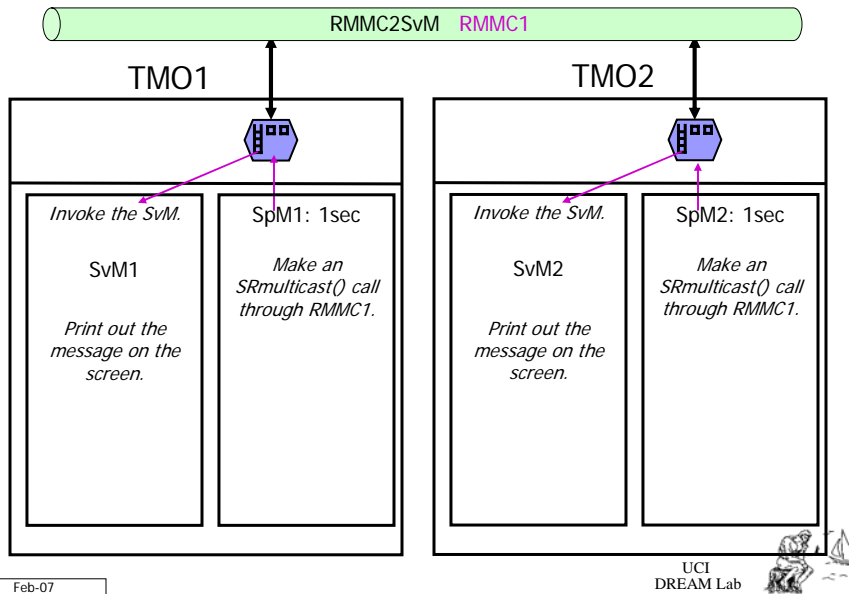
- This example consists of two TMOs which share one RMMC2SvM within one node.
- Each TMO has one SpM and one SvM.
- The SpM in each TMO sends out one SRmulticast () call through the RMMC2SvM at every iteration.
- The SvM in each TMO is bound to the RMMC2SvM channel.
- The SvM in each TMO prints out the received message from on the screen.

Feb-07

UCI  
DREAM Lab



## Example 1: TMO Network Diagram



## Example 1: Source Code

- RMMC\_Triggered\_SvM.h
 

```
#pragma once // included only once by compiler
#include "TMOSL.h"
#pragma comment(lib, "TMOSL.lib") // Include TMOSL.lib

using namespace TMO;

typedef struct _ParamStruct // data structure for application msg passing
{
    __int64 client_req_timestamp;
    TCHAR client_msg[128];
} ParamStruct;
```

Feb-07

UCI  
DREAM Lab



## Example 1: Source Code (cont.)

```
// Application RMMC gate class
class CRMMCGate : public RMMC2SvM_GateBaseClass
{
public:
    CRMMCGate (TCHAR * RMMCname, TCHAR* SvMname) {
        // Register SvM Parameter Message
        build_regist_info_SvMParamMsg (sizeof (ParamStruct));
        // Register SvM
        build_regist_info_SvM (SvMname);
        // Register RMMC2SvM gate
        RegisterRMMC2SvM_gate (RMMCname);
    };
};
```

Feb-07

UCI  
DREAM Lab



## Example 1: Source Code (cont.)

```
// Application TMO class
class CSimpleTMO : public CTMOBase
{
public:
    // constructor
    CSimpleTMO (TCHAR * tmo_name, TCHAR * svm_name,
               tms start_time);
    // destructor
    virtual ~CSimpleTMO () {};

private:
    void          SpM1 ();
    int           SvM1 (ParamStruct *);
    CRMMCGate    RMMC1; // RMMC2SvM-Gate for communication
};
```

Feb-07

UCI  
DREAM Lab



## Example 1: Source Code (cont.)

- RMMC\_Triggered\_SvM.cpp

```
#include "RMMC_Triggered_SvM.h"

// constructor of application TMO class
CSimpleTMO::CSimpleTMO (
    TCHAR * tmo_name, TCHAR * svm_name, tms start_time)
    : RMMC1 (_T("TEST_RMMC_CHANNEL"), svm_name)
    // init RMMC gate
{
```

Feb-07

UCI  
DREAM Lab



## Example 1: Source Code (cont.)

```
// SpM registration
SpM_RegistParam    spm_regist_param;
MicroSec    from = 4 * 1000 * 1000; // This must be later than
// the start time of TMO.

MicroSec    until = 1 * 60 * 60;
            until *= 1000 * 1000; // 1 hour

AAC aac1(
    NULL,
    tm4_DCS_age(from),
    tm4_DCS_age(until),
    1000 * 1000,
    0,
    150 * 1000,
    400 * 1000);
```

Feb-07

UCI  
DREAM Lab



## Example 1: Source Code (cont.)

```
// bind AAC to SpM
spm_regist_param.build_regist_info_AAC (aac1);
// Register RMMC gate as an ODSS
spm_regist_param.build_regist_info_ODSS (RMMC1.GetId (), RW);

// register SpM
RegisterSpM ( (PFSpMBody) SpM1, & spm_regist_param);

// SvM registration
SvM_RegistParam   svm_regist_param;
_tcscopy (svm_regist_param.name, svm_name); // name of SvM
svm_regist_param.GETB = 400 * 1000; // execution time bound
RegisterSvM ( (PFSvMBody) SvM1, &svm_regist_param);
```

Feb-07

UCI  
DREAM Lab



## Example 1: Source Code (cont.)

```
// TMO registration
TMO_RegistParam   tmo_regist_param;
_tcscopy (tmo_regist_param.global_name, tmo_name);
tmo_regist_param.start_time = start_time;
RegisterTMO (& tmo_regist_param);
}
```

Feb-07

UCI  
DREAM Lab



## Example 1: Source Code (cont.)

```
// SpM body
void CSimpleTMO::SpM1 ()
{
    TCHAR buffer[80];
    static SpMCount = 0;
    static ParamStruct param;
    static INT64 seq_num = 0;
    param.client_req_timestamp = seq_num ++;
    _stprintf (buffer, _T("seq = %I64dWttimestamp = %I64dWn"),
              seq_num, GetCurrentDCSage ());
    _tcscpy (param.client_msg, buffer);

    // Send one SvM Parameter Message through RMMC gate
    RMMC1.SRmulticast (&param, sizeof (ParamStruct),
                      tm4_DCS_age ( GetCurrentDCSage () ));
}
```

Prepare SvM  
Parameter Message

Feb-07

UCI  
DREAM Lab



## Example 1: Source Code (cont.)

```
// SvM body
// This SvM is invoked when a SvM Parameter Message is received and
// released
int CSimpleTMO::SvM1 (ParamStruct * param)
{
    // Get the timestamp when the svm parameter message is issued.
    MicroSec Timestamp = GetSRTimestamp ();
    // print out the message on the screen
    TMOSLprintf (_T("time %I64d: %s issued at %I64dWn"), param-
                >client_req_timestamp, param->client_msg, Timestamp);

    return TRUE;
}
```

Feb-07

UCI  
DREAM Lab



## Example 1: Source Code (cont.)

```
// main() function of this application
// This application creates two TMOs in one node.
void main ()
{
    StartTMOengine ();
    // Create TMO1
    CSimpleTMO TMO1 (_T("TMO1"), _T("SvMatTMO1"),
                    tm4_DCS_age (3 * 1000 * 1000));
    // Create TMO2
    CSimpleTMO TMO2 (_T("TMO2"), _T("SvMatTMO2"),
                    tm4_DCS_age (3 * 1000 * 1000));

    MainThrSleep ();
}
```

Feb-07

UCI  
DREAM Lab



## Example 2: Tele-Audio with an RMMC2SvM

- This is a multimedia application designed as a TMO network.
- There are two nodes in the system:
  - One records voice with microphone and sends data to the receiver;
  - The other receives the data from the sender and plays back the sound with a speaker.
- Sender has one SpM which reads sampled data from the audio device and sends data packets to the receiver via SvM Parameter Messages carried through an RMMC2SvM channel.
- Receiver has one SpM and one SvM.  
The SvM is triggered by SvM Parameter Messages.  
It receives audio packets and saves them into a circular buffer ODSS.  
The SpM retrieves audio packets from the circular buffer and writes them into an audio device driver.
- There is one RMMC2SvM channel between sender and receiver which facilitates message exchanges.

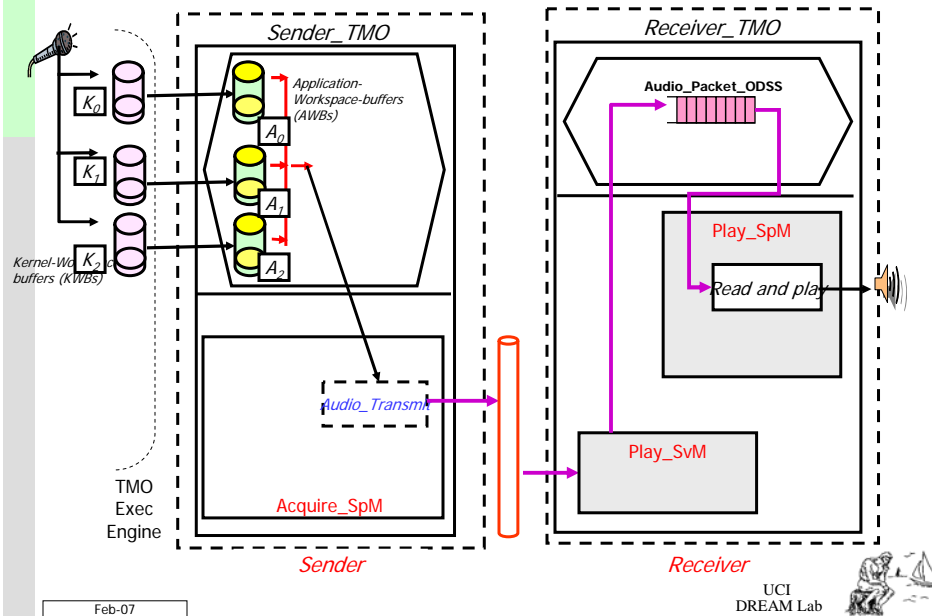
Note: Refer to the lecture slides on standard RMMC for the design details of the tele-audio application.

Feb-07

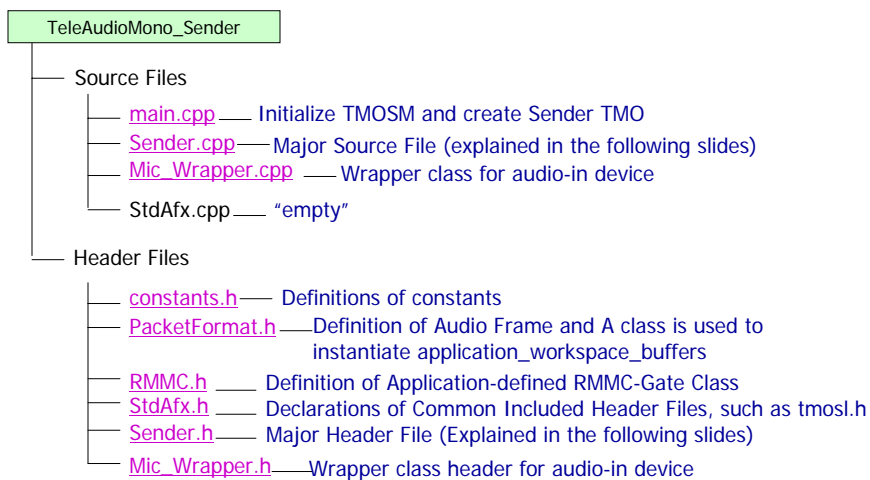
DREAM Lab



## TMO Network for Tele-Audio Application



## Visual Studio Project (VSP) Structures



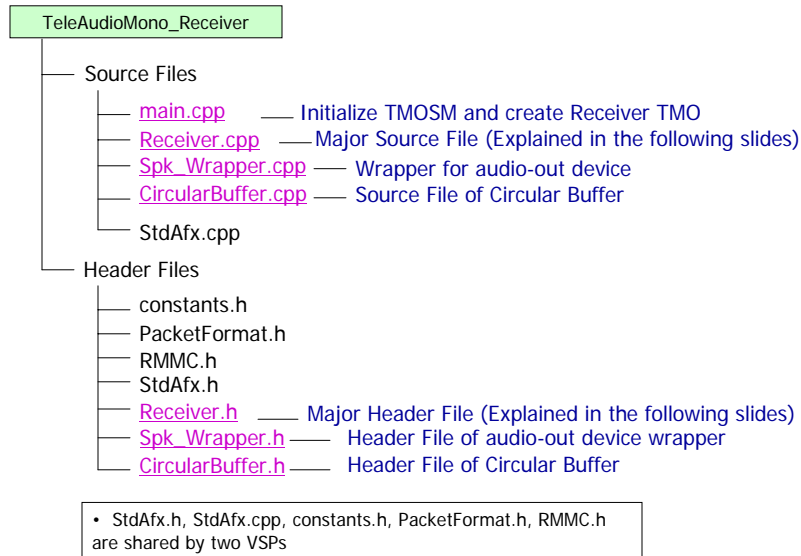
• StdAfx.h, StdAfx.cpp, constants.h, PacketFormat.h, RMMC.h are shared by two VSPs

Feb-07

UCI DREAM Lab



## Visual Studio Project (VSP) Structures

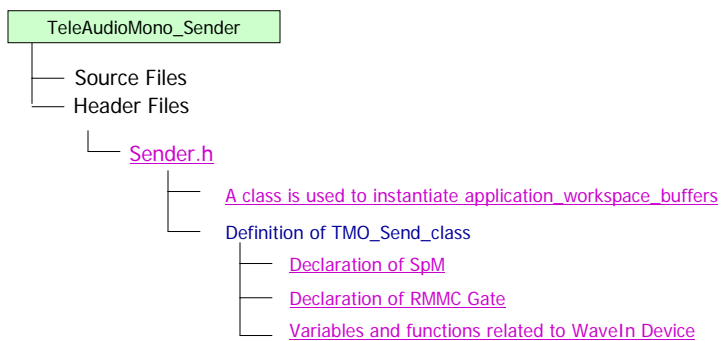


Feb-07

UCI  
DREAM Lab



## TeleAudioMono\_Sender::Sender.h



◀BACK

• StdAfx.h, StdAfx.cpp, constants.h, PacketFormat.h, RMMC.h are shared by two VSPs

Feb-07

UCI  
DREAM Lab



## TeleAudioMono\_Sender::Sender.cpp

TeleAudioMono\_Sender

Header Files  
Source Files

Sender.cpp

Function for SpM Registration  
Function body of Sender\_SpM  
Function body for the constructor of Sender TMO Class

←BACK

Feb-07

UCI  
DREAM Lab



## TeleAudioMono\_Receiver::Receiver.h

TeleAudioMono\_Receiver

Source Files  
Header Files

Receiver.h

Definition of TMO\_Recv\_class  
Declaration of SpM  
Declaration of RMMC Gate  
Declaration of SvM  
Declaration of Audio\_Packet\_ODSS  
Variables and functions related to WaveOut Device

←BACK

Feb-07

UCI  
DREAM Lab



## TeleAudioMono\_Receiver::Receiver.cpp

TeleAudioMono\_Receiver

Header Files

Source Files

[Receiver.cpp](#)

[Function for the registration of SpM of Receiver TMO Class](#)

[Function body of Recv\\_SpM](#)

[Function body for the constructor of Receiver TMO Class](#)

[Function body for the registration of SvM of Receiver TMO Class](#)

[Function body for Recv\\_SvM](#)

[←BACK](#)

Feb-07

UCI  
DREAM Lab



## TeleAudio – Sender Node

Feb-07

UCI  
DREAM Lab



# constants.h

```
#ifndef constants_h  
#define constants_h
```

// SpM-related & SvM-related constants

◀BACK

```
#define WARMUP_DELAY_SECS 5  
#define SYSTEM_LIFE_HOURS 24  
#define DEALINE_MSEC_RECEIVER_SVM 3  
  
#define SENDER_SPM_PERIOD 27  
#define RECEIVER_SPM_PERIOD 27  
#define SENDER_SPM_DEADLINE 18  
#define RECEIVER_SPM_DEADLINE 18  
#define LASTEST_SPM_START_TIME 10
```

Feb-07

UCI  
DREAM Lab



# constants.h (cont')

```
#define SAMPLE_RATE 8000  
#define SAMPLE_SIZE 1  
#define CHANNEL_NUM 1 // mono or stereo  
#define WAVEIN_AUDIO_BLOCK_SIZE ((CHANNEL_NUM *  
SAMPLE_SIZE * SAMPLE_RATE * SENDER_SPM_PERIOD)/1000)  
// WaveIn audio block/buffer size.  
#define IN_BUFFER 3 // The number of WaveIn buffers.  
#define WAVEOUT_AUDIO_BLOCK_SIZE ((CHANNEL_NUM * SAMPLE_SIZE  
* SAMPLE_RATE * RECEIVER_SPM_PERIOD)/1000)  
// WaveOut audio block/buffer size.  
#define OUT_BUFFER 3 // The number of WaveOut buffers.  
#define AUDIO_TSD 50 // Audio Target Streaming Delay
```

// WaveIn&WaveOut device-related constants

Feb-07

UCI  
DREAM Lab



# PacketFormat.h

```
#ifndef __PACKET_FORMAT_H__
#define __PACKET_FORMAT_H__
```

◀BACK

```
#include "StdAfx.h"
#include "constants.h"
using namespace TMO;
```

// The data structure of a frame

```
typedef struct
{
    unsigned int    aFrameID; // ID of a frame
    MicroSec        aICT;     // Imaginary Capture Timestamp of a frame
    MicroSec        aTPT;     // Target Play Timestamp of a frame
    MicroSec        aSend;    // Sending Timestamp of a frame
    MicroSec        aRecv;    // Receiving Timestamp of a frame (recorded at the
                             // receiver side
    unsigned char   aData [WAVEIN_AUDIO_BLOCK_SIZE]; // Audio Data
} SAudioFrame;
```

```
#endif
```

Feb-07

UCI  
DREAM Lab



# RMMC.h

```
#ifndef _RMMC_H_
#define _RMMC_H_
#include "stdafx.h"
#include "PacketFormat.h"
```

◀BACK

```
class RMMCReceiverGateClass: public RMMC2SvM_GateBaseClass
{
public:
    RMMCReceiverGateClass (TCHAR* RMMC_name)
    {
        // Register SvM
        build_regist_info_SvM (_T("Audio_Recv_SvM"));
        // RMMC gate Registration
        RegisterRMMC2SvM_gate (RMMC_name);
    };
};
```

// RMMCReceiverGateClass is  
inherited from  
RMMC2SvM\_GateBaseClass

Feb-07

UCI  
DREAM Lab



## RMMC.h (cont')

```
class RMMCSenderGateClass: public RMMC2SvM_GateBaseClass
{
public:
    RMMCSenderGateClass (TCHAR* RMMC_name) // RMMCSenderGateClass is
    {                                       inherited from
    // build SvM Parameter message info   RMMC2SvM_GateBaseClass
    build_regist_info_SvMParamMsg (sizeof (SAudioFrame));
    // RMMC gate Registration
    RegisterRMMC2SvM_gate (RMMC_name);
    };
};
#endif // _RMMC_H_
```

◀BACK

Feb-07

UCI  
DREAM Lab



## Mic\_Wrapper.h

```
#ifndef __MIC_WRAPPER_H__
#define __MIC_WRAPPER_H__

#include "StdAfx.h"
#include "constants.h"
#pragma pack(4)

using namespace TMO;

// The class is for microphone, which is inherited from ODSSBaseClass
class Mic_Wrapper_Class : public ODSSBaseClass <Mic_Wrapper_Class>
{
public:
    Mic_Wrapper_Class () {};
    ~Mic_Wrapper_Class () {};
```

◀BACK

Feb-07

UCI  
DREAM Lab



## Mic\_Wrapper.h

```
// Initialize the WaveIn device.
void          OpenMic ();
BOOL         IsMicStart ()    { return AudioIn_start; }
MMRESULT     Read (int buffer_number);
char*        GetBufferData (int buffer_number) { return AWB[buffer_number];}

private:
char AWB [IN_BUFFER] [WAVEIN_AUDIO_BLOCK_SIZE];
WAVEFORMATEX wave_format; // WaveIn Device Setting Parameter
HWAVEIN       wavein_handle; // The handle of WaveIn device.
WAVEHDR       wavein_wave_hdr [IN_BUFFER];
// The headers for each WaveIn buffer.

int           wavein_open_result; // Status flag on WaveIn open operation.
BOOL         AudioIn_start; // Flag for microphone start.

};
#endif
```

Feb-07

UCI  
DREAM Lab



## Sender.h

```
#ifndef __SENDER_H__
#define __SENDER_H__

#include "StdAfx.h"
#include "PacketFormat.h"
#include "RMMC.h"
#include "Mic_Wrapper.h"

using namespace TMO;
```

Feb-07

UCI  
DREAM Lab



## Sender.h (con't)

```
// class : TMO_Class
class TMO_Send_Class: public CTMOBase
{
private:
    int                Sender_SpM ();    // SpM body
    RMMCSenderGateClass RMMC_1;    // RMMC gate
    Mic_Wrapper_Class  m_MIC;    // wrapper object for audio-in device

    // Local functions
    void                Sender_SpM_Register_Init ();    // SpM Initialization Method
```

◀BACK

Feb-07

UCI  
DREAM Lab



## Sender.h (con't)

```
__int64    A_nMessageID;    //Frame counter
__int64    SpM_iteration;    // Counter for SpM iteration

unsigned char    LastBufferCount;

public:
    TMO_Send_Class (TCHAR *, tms &);
};
#endif
```

// Auxiliary variables used by  
// Sender\_SpM () only

// Keep tracking of the buffer  
// used lately & used by  
// Sender\_SpM () only

◀BACK

Feb-07

UCI  
DREAM Lab



# stdAfx.h

```
// stdafx.h : include file for standard system include files,  
// or project specific include files that are used frequently, but  
// are changed infrequently
```

◀BACK

// The following part in this slide is automatically generated by Visual Studio

```
#if !defined  
    (AFX_STDAFX_H__B1409B15_75F8_11D3_BF6B_00A0CC3FBC6E__INCLUDED_)  
#define AFX_STDAFX_H__B1409B15_75F8_11D3_BF6B_00A0CC3FBC6E__INCLUDED_  
  
#if _MSC_VER > 1000  
#pragma once  
#endif // _MSC_VER > 1000
```

Feb-07

UCI  
DREAM Lab



# stdAfx.h (cont')

// Additional headers program requires

```
//#define AUDIO_RMMC_to_SvM  
#include <tchar.h>  
//#define NOTPT  
#include <windows.h>  
#include <vfw.h>
```

// TMOsl header

```
#include <tmosl.h>  
#pragma comment ( lib, "winmm")
```

// The following part in this slide is automatically generated by Visual Studio

```
//{{AFX_INSERT_LOCATION}}  
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.  
  
#endif // !defined (AFX_STDAFX_H__B1409B15_75F8_11D3_BF6B_00A0CC3FBC6E__INCLUDED_)
```

Feb-07

UCI  
DREAM Lab



## main() (Sender)

```
#include "stdafx.h"
#include "TMO_Send.h"
void main ()
{
    // Start TMO support Middleware
    StartTMOengine ();
    tms    start_time = tm4_DCS_age ( 5*1000*1000);

    TMO_Send_Class* pTMO_Send = new
        TMO_Send_Class (_T("TMO_Send"), start_time);

    // Main thread goes to sleep and TMO SM takes the control
    MainThrSleep ();
}
```

◀BACK

An alternative way to instantiate an TMO obj:

```
TMO_Send_Class SenderTMO
(_T("TMO_Send"), start_time);
```

Feb-07

UCI  
DREAM Lab



## Mic\_Wrapper.cpp

```
#include "Mic_Wrapper.h"

// read kernel buffer
MMRESULT Mic_Wrapper_Class::Read (int buffer_number)
{
    MMRESULT    result;
    WAVEHDR     *pWaveHdr;

    pWaveHdr = (WAVEHDR*) & wavein_wave_hdr [buffer_number];
    pWaveHdr->dwBufferLength = WAVEOUT_AUDIO_BLOCK_SIZE;

    // Send the input buffer to the input device.
    result = waveInAddBuffer (wavein_handle,
                              pWaveHdr,
                              sizeof (WAVEHDR));

    return result;
}
```

◀BACK

Feb-07

UCI  
DREAM Lab



## Mic\_Wrapper.cpp

```
/*
 * AudioIn_Open opens the waveform-audio input device for reading input data.
 * It also initializes the buffers used for receiving data from the audio
 * input device.
 */
void Mic_Wrapper_Class::OpenMic ()
{
    // Settings for the WaveIn device.
    memset ( (void *) & wave_format, 0, sizeof (wave_format));
    wave_format.wFormatTag      = WAVE_FORMAT_PCM;
    wave_format.nChannels      = CHANNEL_NUM;
    wave_format.nSamplesPerSec = SAMPLE_RATE;
    wave_format.nAvgBytesPerSec = SAMPLE_RATE;
    wave_format.nBlockAlign    = 1;
    wave_format.wBitsPerSample = 8 * SAMPLE_SIZE;
    wave_format.cbSize         = 0;
```

Feb-07

UCI  
DREAM Lab



## Mic\_Wrapper.cpp

```
/* Open the waveform-audio input device.*/
wavein_open_result = waveInOpen (
    (HWAVEIN *) & wavein_handle,
    WAVE_MAPPER,
    (WAVEFORMATEX *) & wave_format,
    (DWORD) 0,
    (DWORD) 0,
    CALLBACK_NULL);

/* If device open operation fails, print error message.*/
if (wavein_open_result != MMSYSERR_NOERROR)
{
    wavein_handle = NULL;
    TMOStprintf (_T("*** WaveIn device open failed.***\n"));
    return ;
}
```

Feb-07

UCI  
DREAM Lab



## Mic\_Wrapper.cpp

```
/* Initialize the waveform-audio input buffer fields.*/
for (int i = 0; i < IN_BUFFER; i++)
{
    wavein_wave_hdr[i].lpData          = (char *) AWB [i];
    wavein_wave_hdr[i].dwBufferLength  = WAVEIN_AUDIO_BLOCK_SIZE;
    wavein_wave_hdr[i].dwUser          = 1;
    wavein_wave_hdr[i].dwFlags         = 0;

    /* Prepare a waveform-audio input.*/
    waveInPrepareHeader (wavein_handle, & wavein_wave_hdr[i], sizeof
(WAVEHDR) );
}

/* Starts input on the given waveform-audio input device.*/
MMRESULT result = waveInStart (wavein_handle);
```

Feb-07

UCI  
DREAM Lab



## Mic\_Wrapper.cpp

```
/* Check for the return code.*/
switch (result) {
    case MMSYSERR_INVALIDHANDLE:
        TMOSLprintf ( _T("waveInStart: MMSYSERR_INVALIDHANDLE"));
        break;
    case MMSYSERR_NODRIVER:
        TMOSLprintf ( _T("waveInStart: MMSYSERR_NODRIVER"));
        break;
    case MMSYSERR_NOMEM:
        TMOSLprintf ( _T("waveInStart: MMSYSERR_NOMEM"));
        break;
    case MMSYSERR_NOERROR:
        break;
}
AudioIn_start = true;
return;
```

Feb-07

UCI  
DREAM Lab



# Sender.cpp

```
#include <iostream>

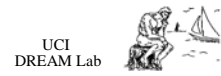
#include "Sender.h"
#include "mmreg.h"
#include "Mmsystem.h"

using namespace TMO;

void TMO_Send_Class::Sender_SpM_Register_Init ()
{
    LastBufferCount = 0;
```



Feb-07



# Sender.cpp (con't)

```
A_nMessageID = 0;
SpM_RegistParam Sender_SpM_spec;
```



//specify time window for SpM

```
MicroSec from = (MicroSec) WARMUP_DELAY_SECS * 1000 * 1000;
MicroSec until = (MicroSec) SYSTEM_LIFE_HOURS * 60 * 60 * 1000 * 1000;
MicroSec every = (MicroSec) SENDER_SPM_PERIOD * 1000;
MicroSec est = 0;
MicroSec lst = est + LASTEST_SPM_START_TIME * 1000;
MicroSec by = SENDER_SPM_DEADLINE * 1000;
```

Fill parameters for AAC

Feb-07



## Sender.cpp (con't)

```
AAC aac1 (  
    NULL, // null for candidate aac label  
    tm4_DCS_age ((MicroSec)(WARMUP_DELAY_SECS * 1000 * 1000)) ,  
    tm4_DCS_age ((MicroSec)(20 * 60 * 1000 * 1000)),  
    every,  
    est,  
    lst,  
    by  
); // The instantiation of AAC obj
```

```
Sender_SpM_spec.build_regist_info_AAC (aac1); // Sender SpM  
// Register RMMC gate as an ODSS Registration  
Sender_SpM_spec.build_regist_info_ODSS (RMMC_1.GetId(), RW);  
// Register Application-Workspace-buffer as ODSS  
Sender_SpM_spec.build_regist_info_ODSS (m_MIC.GetId(), RW);  
// register SpM  
RegisterSpM ((PFSpMBody) Sender_SpM, & Sender_SpM_spec);  
}
```

Feb-07

UCI  
DREAM Lab



## Sender.cpp (con't)

```
int TMO_Send_Class::Sender_SpM ()
```

```
{
```

```
MicroSec        official_release_time;  
MMRESULT        result;  
int              i;  
SAudioFrame     AudioFrame;
```

// Temporary  
variables used in  
SpM

◀BACK

Feb-07

UCI  
DREAM Lab



## Sender.cpp (con't)

// If the waveform-audio input devices has been initialized.

```
if ( m_MIC.IsMicStart () )
```

```
{
```

```
    for (i = 0; i < IN_BUFFER; i++)
    {
        if (LastBufferCount + 1 == IN_BUFFER)
        {
            result = m_MIC.Read (LastBufferCount + 1 - IN_BUFFER);
        }
        else {
            result = m_MIC.Read (LastBufferCount + 1);
        }
    }
```

// Get an audio frame from a buffer

◀BACK

Feb-07

UCI  
DREAM Lab



## Sender.cpp (con't)

```
if (result == MMSYSERR_NOERROR) {
```

```
    if (LastBufferCount + 1 == IN_BUFFER) {
        memcpy ( AudioFrame.aData, m_MIC.GetBufferData ( LastBufferCount + 1 -
            IN_BUFFER), WAVEIN_AUDIO_BLOCK_SIZE);
        LastBufferCount = LastBufferCount + 1 - IN_BUFFER;
    } else {
        memcpy ( AudioFrame.aData, m_MIC.GetBufferData ( LastBufferCount + 1),
            WAVEIN_AUDIO_BLOCK_SIZE);
        LastBufferCount++;
    }
    AudioFrame.aFrameID = A_nMessageID;
    TMOSLprintf ( _T("From Buffer %d for %dWn"), LastBufferCount, A_nMessageID);
    official_release_time = GetCurrentDCSage () + 100 * 1000;
    AudioFrame.aSend = GetCurrentDCSage ();
```

// Prepare audio frame to be sent out

Feb-07

UCI  
DREAM Lab



## Sender.cpp (con't)

```
        if(!RMMC_1.SRmulticast ( (void *) &AudioFrame, sizeof (SAudioFrame),
                                official_release_time) == SUCCESS)
            TMOSLprintf (_T("Fail to send Audio Signal %d SuccessfullyWn"),
                        AudioFrame.aFrameID);
        else
            TMOSLprintf (_T("Success to Send Audio Signal %dWn"),
                        AudioFrame.aFrameID);
        A_nMessageID++;
    }
    else
        break; // Send out the frames
}
return 1;
}
```

Feb-07

UCI  
DREAM Lab



## Sender.cpp (con't)

```
TMO_Send_Class::TMO_Send_Class (TCHAR* TMO_name, tms & start_time)
:RMMC_1 (_T("RMMC_1"))
{
    /* Initialize the WaveIn device.*/
    m_MIC.OpenMic ();

    //register Video_SpM
    Sender_SpM_Register_Init ();

    //register TMO
    TMO_RegistParam TMO_Send_spec;
    _tcscpy (TMO_Send_spec.global_name, TMO_name);
    TMO_Send_spec.start_time = start_time;
    RegisterTMO (&TMO_Send_spec);
}
```

Feb-07

UCI  
DREAM Lab



## TeleAudio – Receiver Node

Feb-07

UCI  
DREAM Lab



## CircularBuffer.h

```
#ifndef __CIRCULAR_BUFFER_H__
#define __CIRCULAR_BUFFER_H__

#include "StdAfx.h"
#include "constants.h"
#include "PacketFormat.h"
using namespace TMO;
#define AUDIO_RECEIVED_BUFFER_NUMBER 50
```

◀BACK

// Circular Buffer Class which is used to instantiate circular buffers in the receiver side.

```
class Temp_Audio_Class:: public ODSSBaseClass <Temp_Audio_Class>
{
private:
    // Circular buffer is built on top of an array
    SAudioFrame    AudioFrame[AUDIO_RECEIVED_BUFFER_NUMBER];
    unsigned int    AudioBufferHead; // Current Occupied Buffer Header
    unsigned int    AudioBufferTail; // Current Occupied Buffer Tail
    unsigned int    LastAction; // 0:Write, 1:Read, 2:Start. Record the last operation
                    // on this circular buffer
```

Feb-07

UCI  
DREAM Lab



## CircularBuffer.h (cont')

```
public:
    Temp_Audio_Class () : AudioBufferHead(0), AudioBufferTail(0), LastAction(2) {};
    ~Temp_Audio_Class () {};

    bool WriteAudioFrame (SAudioFrame*); // Write a frame into the circular buffer
    bool CopyFrame (SAudioFrame*); // Retrieve a frame from the circular buffer

    bool IsAudioBufferEmpty (); // Some auxiliary methods
    MicroSec GetTPTofAudioFrame (unsigned int);
    MicroSec GetICTofAudioFrame (unsigned int);
    MicroSec GetSendTimeofAudioFrame (unsigned int);
    MicroSec GetRecvTimeofAudioFrame (unsigned int);
    unsigned int GetFrameIDofAudioFrame (unsigned int);
    unsigned int GetAudioBufferHead ();
    unsigned int GetAudioBufferTail ();
    unsigned int GetBufferSize ();
};
```

Feb-07

UCI  
DREAM Lab



## Spk\_Wrapper.h

```
#ifndef __SPK_WRAPPER_H__
#define __SPK_WRAPPER_H__

#include "StdAfx.h"
#include "constants.h"
#pragma pack(4)

using namespace TMO;

// The class is for speaker, which is inherited from ODSSBaseClass
class Spk_Wrapper_Class : public ODSSBaseClass <Spk_Wrapper_Class>
{
public:
    Spk_Wrapper_Class () {};
    ~Spk_Wrapper_Class () {};
```

Feb-07

UCI  
DREAM Lab



# Spk\_Wrapper.h

```
// Initialize the WaveOut device.
void      OpenSpk ();
BOOL      IsSpkStart () { return AudioOut_start;}
void      Play (char * waveout_dataPtr, int waveout_size);
          // Method for Playing Audio Frames

private:
char AWB [OUT_BUFFER] [WAVEIN_AUDIO_BLOCK_SIZE];

WAVEFORMATEX wave_format; // WaveOut Device Setting Parameter
HWAVEOUT      waveout_handle; // The handle of WaveOut device.
WAVEHDR       waveout_wave_hdr[OUT_BUFFER];
              // The headers for each WaveOut buffer.

int           waveout_open_result; // Status flag on WaveOut open operation.
BOOL          AudioOut_start;     // Flag for speaker start.
};
#endif
```

Feb-07

UCI  
DREAM Lab



# RMMC.h

```
#ifndef _RMMC_H_
#define _RMMC_H_
#include "stdafx.h"
#include "PacketFormat.h"

class RMMCReceiverGateClass: public RMMC2SvM_GateBaseClass
{
public:
    RMMCReceiverGateClass (TCHAR* RMMC_name)
    {
        // Register SvM
        build_regist_info_SvM (_T("Audio_Recv_SvM"));
        // RMMC gate Registration
        RegisterRMMC2SvM_gate (RMMC_name);
    };
};
```

Feb-07

UCI  
DREAM Lab



## RMMC.h (cont')

```
class RMMCSenderGateClass: public RMMC2SvM_GateBaseClass
{
public:
    RMMCSenderGateClass (TCHAR* RMMC_name) // RMMCSenderGateClass is
    {                                       inherited from
    // build SvM Parameter message info   RMMC2SvM_GateBaseClass
    build_regist_info_SvMParamMsg (sizeof (SAudioFrame));
    // RMMC gate Registration
    RegisterRMMC2SvM_gate (RMMC_name);
    };
};
#endif // _RMMC_H_
```

◀BACK

Feb-07

UCI  
DREAM Lab



## Receiver.h

```
#ifndef __RECEIVER_H__
#define __RECEIVER_H__

#include "PacketFormat.h"
#include "CircularBuffer.h"
#include "RMMC.h"
#include "Spk_Wrapper.h"

using namespace TMO;
```

◀BACK

Feb-07

UCI  
DREAM Lab



## Receiver.h (con't)

```
// class : TMO_Class
class TMO_Recv_Class: public CTMOBase
{
private:
    int          Recv_SpM (); // SpM method
    void         Audio_Recv_SvM (SAudioFrame*); // SvM method

    RMMCReceiverGateClass RMMC_1;          // RMMC gate
    Spk_Wrapper_Class     m_SPK;          //Speak Wrapper
    Temp_Audio_Class      Audio_Packet_ODSS; //Audio Buffer shared by SvM & SpM
    // Local functions
    void          Recv_SpM_Register_Init (); // SpM initialization method
    void          Audio_Recv_SvM_Register_Init (); // SvM initialization method
public:
    TMO_Recv_Class (TCHAR *, tms &);
};
#endif
```

◀BACK

Feb-07

UCI  
DREAM Lab



## stdAfx.h

```
// stdafx.h : include file for standard system include files,
// or project specific include files that are used frequently, but
// are changed infrequently
```

◀BACK

// The following part in this slide is automatically generated by Visual Studio

```
#if !defined
(AFX_STDAFX_H__B1409B15_75F8_11D3_BF6B_00A0CC3FBC6E__INCLUDED_)
#define AFX_STDAFX_H__B1409B15_75F8_11D3_BF6B_00A0CC3FBC6E__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
```

Feb-07

UCI  
DREAM Lab



## stdAfx.h (cont')

```
// Additional headers program requires
```

```
//#define AUDIO_RMMC_to_SvM
#include <tchar.h>
//#define NOTPT
#include <windows.h>
#include <vfw.h>
```

```
// TMOsl header
```

```
#include <tmosl.h>
#pragma comment ( lib, "winmm")
```

```
// The following part in this slide is automatically generated by Visual Studio
```

```
//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

#endif // !defined (AFX_STDAFX_H__B1409B15_75F8_11D3_BF6B_00A0CC3FBC6E__INCLUDED_)
```

Feb-07

UCI  
DREAM Lab



## main() (Receiver)

```
#include "stdafx.h"
#include "TMO_Recv.h"
```

◀BACK

```
void main ()
{
```

```
    // Start TMO support Middleware
    StartTMOengine ();
```

```
    tms    start_time = tm4_DCS_age ( 5*1000*1000);
```

```
    TMO_Recv_Class* pTMO_Recv = new TMO_Recv_Class (_T("TMO_Recv"),
    start_time);
```

```
    // Main thread goes to sleep and TMOsl takes the control
    MainThrSleep ();
```

```
}
```

Feb-07

UCI  
DREAM Lab



# CircularBuffer.cpp

```
#include "CircularBuffer.h"
bool Temp_Audio_Class::WriteAudioFrame ( SAudioFrame* pPara )
{
    if (AudioBufferHead == AudioBufferTail) {
        if (LastAction==1||LastAction==2){
            AudioFrame [AudioBufferHead].aICT = pPara->aICT;
            AudioFrame [AudioBufferHead].aTPT = pPara->aTPT;
            AudioFrame [AudioBufferHead].aFrameID = pPara->aFrameID;
            AudioFrame [AudioBufferHead].aSend = pPara->aSend;
            AudioFrame [AudioBufferHead].aRecv = pPara->aRecv;
            memcpy (          AudioFrame[AudioBufferHead].aData,
                pPara->aData,
                WAVEOUT_AUDIO_BLOCK_SIZE);
            if ((AudioBufferHead+1) == AUDIO_RECEIVED_BUFFER_NUMBER){
                AudioBufferHead = AudioBufferHead+1-
                    AUDIO_RECEIVED_BUFFER_NUMBER;
            }
            else AudioBufferHead++;
            LastAction = 0;
        }
        else return false;
    }
}
```

// Insert an audio  
frame into the  
circular buffer if  
the buffer is  
empty

Feb-07

UCI  
DREAM Lab



# CircularBuffer.cpp (con't)

```
else {
    AudioFrame [AudioBufferHead].aICT = pPara->aICT;
    AudioFrame [AudioBufferHead].aTPT = pPara->aTPT;
    AudioFrame [AudioBufferHead].aFrameID = pPara->aFrameID;
    AudioFrame [AudioBufferHead].aSend = pPara->aSend;
    AudioFrame [AudioBufferHead].aRecv = pPara->aRecv;
    memcpy ( AudioFrame [AudioBufferHead].aData,
        pPara->aData,
        WAVEOUT_AUDIO_BLOCK_SIZE);
    if ((AudioBufferHead+1) == AUDIO_RECEIVED_BUFFER_NUMBER) {
        AudioBufferHead = AudioBufferHead+1-
            AUDIO_RECEIVED_BUFFER_NUMBER;
    }
    else AudioBufferHead++;
    LastAction = 0;
}
return true;
}
```

// Insert an audio  
frame into the  
circular buffer if  
the buffer is not  
empty

Feb-07

UCI  
DREAM Lab



## CircularBuffer.cpp (con't)

```
bool Temp_Audio_Class::CopyFrame ( SAudioFrame* pPara ) {
    if (AudioBufferHead == AudioBufferTail) {
        if (LastAction==0) {
            pPara->aICT = AudioFrame[AudioBufferTail].aICT;
            pPara->aTPT = AudioFrame[AudioBufferTail].aTPT;
            pPara->aFrameID = AudioFrame[AudioBufferTail].aFrameID;
            pPara->aSend = AudioFrame[AudioBufferTail].aSend;
            pPara->aRecv = AudioFrame[AudioBufferTail].aRecv;
            memcpy(pPara->aData, AudioFrame [AudioBufferTail].aData,
                WAVEOUT_AUDIO_BLOCK_SIZE);
            if(AudioBufferTail+1 == AUDIO_RECEIVED_BUFFER_NUMBER)
            {
                AudioBufferTail = AudioBufferTail+1-
                    AUDIO_RECEIVED_BUFFER_NUMBER;
            }
            else AudioBufferTail++;

            LastAction = 1;

            return true;
        }
    }
}
```

// Get the first audio frame from the circular buffer if the buffer is full.

Feb-07

UCI  
DREAM Lab



## CircularBuffer.cpp (con't)

```
else {
    pPara->aICT = AudioFrame [AudioBufferTail].aICT;
    pPara->aTPT = AudioFrame [AudioBufferTail].aTPT;
    pPara->aFrameID = AudioFrame [AudioBufferTail].aFrameID;
    pPara->aSend = AudioFrame [AudioBufferTail].aSend;
    pPara->aRecv = AudioFrame [AudioBufferTail].aRecv;

    memcpy(pPara->aData, AudioFrame [AudioBufferTail].aData, WAVEOUT_AUDIO_BLOCK_SIZE);

    if(AudioBufferTail+1 == AUDIO_RECEIVED_BUFFER_NUMBER)
    {
        AudioBufferTail = AudioBufferTail+1-AUDIO_RECEIVED_BUFFER_NUMBER;
    }
    else AudioBufferTail++;

    LastAction = 1;

    return true;
}
return false;
}
```

// Get the first audio frame from the circular buffer if the buffer is not full.

Feb-07

UCI  
DREAM Lab



## CircularBuffer.cpp (con't)

```
bool Temp_Audio_Class::IsAudioBufferEmpty ()
{
    if (AudioBufferTail == AudioBufferHead)
    {
        if (LastAction == 1 || LastAction == 2) return true;
        else if (LastAction == 0) return false;
    }
    else return false;
}

MicroSec Temp_Audio_Class::GetTPTofAudioFrame (unsigned int Index)
{
    return AudioFrame[Index].aTPT;
}

unsigned int Temp_Audio_Class::GetFrameIDofAudioFrame (unsigned int Index)
{
    return AudioFrame[Index].aFrameID;
}
```

// The function bodies of auxiliary methods

Feb-07

UCI  
DREAM Lab



## CircularBuffer.cpp (con't)

```
unsigned int Temp_Audio_Class::GetAudioBufferHead () {
    return AudioBufferHead;
}

unsigned int Temp_Audio_Class::GetAudioBufferTail () {
    return AudioBufferTail;
}

unsigned int Temp_Audio_Class::GetBufferSize () {
    return AUDIO_RECEIVED_BUFFER_NUMBER;
}

MicroSec Temp_Audio_Class::GetICTofAudioFrame (unsigned int Index) {
    return AudioFrame[Index].aICT;
}

MicroSec Temp_Audio_Class::GetSendTimeofAudioFrame (unsigned int Index) {
    return AudioFrame[Index].aSend;
}

MicroSec Temp_Audio_Class::GetRecvTimeofAudioFrame (unsigned int Index){
    return AudioFrame[Index].aRecv;
}
```

// The function bodies of auxiliary methods

Feb-07

UCI  
DREAM Lab



# Spk\_Wrapper.cpp

```
#include "Spk_Wrapper.h"
```

```
// Function for playing audio frames
```

```
// Play out audio frames
```

◀BACK

```
void Spk_Wrapper_Class::Play (char *waveout_dataPtr, int waveout_size)
{
    int buf = 0;
    if (!waveout_open_result) return;
    int k=0;
    // get a new free WAVEHDR buffer
    for (buf = 0; buf < OUT_BUFFER; buf++)
    {
        DWORD flag = waveout_wave_hdr[buf].dwFlags;
        // check if a wave out buffer is empty
        if (flag== (WHDR_DONE | WHDR_PREPARED) || flag == WHDR_PREPARED) {
            memcpy ( waveout_wave_hdr[buf].lpData,(char *) (waveout_dataPtr), waveout_size);
            waveout_wave_hdr[buf].dwUser = 1;
            waveout_wave_hdr[buf].dwBufferLength = waveout_size;
            int res = waveOutWrite (waveout_handle, &waveout_wave_hdr[buf], sizeof (WAVEHDR));
```

Feb-07

UCI  
DREAM Lab



# Spk\_Wrapper.cpp (con't)

```
switch (res) {
    case MMSYSERR_INVALIDHANDLE:
        TMOSLprintf(_T("waveOutWrite MMSYSERR_INVALIDHANDLEWn"));
        break;
    case MMSYSERR_NODRIVER:
        TMOSLprintf(_T("waveOutWrite MMSYSERR_NODRIVERWn"));
        break;
    case MMSYSERR_NOMEM:
        TMOSLprintf(_T("waveOutWrite MMSYSERR_NOMEMWn"));
        break;
    case WAVERR_UNPREPARED:
        TMOSLprintf(_T("waveOutWrite WAVERR_UNPREPAREDWn"));
        break;
    case MMSYSERR_NOERROR:
        break;
}
```

```
} // if
```

```
} // for
```

```
} Feb-07
```

```
// All kinds of error messages when  
failing to play out audio frames
```

UCI  
DREAM Lab



## Spk\_Wrapper.cpp (con't)

```
/*  
 * AudioOut_Open opens the waveform-audio output device for playback.  
 * It also initializes the buffers used for sending data to the audio  
 * output device.  
 */  
  
void Spk_Wrapper_Class::OpenSpk ()  
{  
    int i;  
    int result;  
    WAVEFORMATEX waveout_wave_format;
```

◀BACK

Feb-07

UCI  
DREAM Lab



## Spk\_Wrapper.cpp (con't)

```
/* Settings for waveform-audio output device.*/
```

```
memset ( (void*) & waveout_wave_format, 0, sizeof (waveout_wave_format));  
waveout_wave_format.wFormatTag = WAVE_FORMAT_PCM;  
waveout_wave_format.nChannels = CHANNEL_NUM;  
waveout_wave_format.nSamplesPerSec = SAMPLE_RATE;  
waveout_wave_format.nAvgBytesPerSec = SAMPLE_RATE;  
waveout_wave_format.nBlockAlign = 1;  
waveout_wave_format.wBitsPerSample = 8 * SAMPLE_SIZE;  
waveout_wave_format.cbSize = 0;
```

```
/* Open the waveform-audio output device.*/
```

```
result = waveOutOpen (  
    (HWAVEOUT *) &waveout_handle,  
    WAVE_MAPPER,  
    (WAVEFORMATEX *) &waveout_wave_format,  
    (DWORD) 0,  
    (DWORD) 0,  
    CALLBACK_NULL );
```

Feb-07

UCI  
DREAM Lab



## Spk\_Wrapper.cpp (con't)

```
/* If device open operation fails, print error message.*/  
if (result != MMSYSERR_NOERROR) {  
    waveout_handle = NULL;  
    TMOSLprintf (_T("WaveOut device open failed.Wn"));  
    return ;  
}  
  
/* Initialize the waveform-audio output buffer fields.*/  
for (i = 0; i < OUT_BUFFER; i++) {  
    waveout_wave_hdr[i].lpData = AWB [i];  
    waveout_wave_hdr[i].dwBufferLength = WAVEOUT_AUDIO_BLOCK_SIZE;  
    waveout_wave_hdr[i].dwUser = 0;  
    waveout_wave_hdr[i].dwFlags = 0L;  
    /* Prepare a waveform-audio data block for playback.*/  
    waveOutPrepareHeader (waveout_handle, &waveout_wave_hdr[i], sizeof (WAVEHDR));  
}  
AudioOut_start = 1; // Indicating the start-up of WaveOut device  
return;  
}
```

Feb-07

UCI  
DREAM Lab



## Receiver.cpp

```
#include <iostream>  
#include "Reciever.h"  
#include "mmreg.h"  
#include "Mmsystem.h"
```

◀BACK

Feb-07

UCI  
DREAM Lab



## Receiver.cpp (con't)

```
// Initialize SpM
void TMO_Recv_Class::Recv_SpM_Register_Init ()
{
    SpM_RegistParam Recv_SpM_spec;
```

◀BACK

```
MicroSec every = (RECEIVER_SPM_PERIOD) * 1000;
MicroSec est = 0;
MicroSec lst = est + (LASTEST_SPM_START_TIME) * 1000;
MicroSec by = (RECEIVER_SPM_DEADLINE) * 1000;
```

Fill parameters for AAC

Feb-07

UCI  
DREAM Lab



## Receiver.cpp (con't)

```
AAC * aac1 = new AAC (
    NULL, // null for candidate aac label
    tm4_DCS_age (9 * 500 * 1000),
    tm4_DCS_age ( (MicroSec) (20 * 60 * 1000 * 1000)),
    every,
    est,
    lst,
    by
);
Recv_SpM_spec.build_regist_info_AAC (*aac1);
```

The instantiation of  
AAC object

```
//register RMMC gate as an ODSS
Recv_SpM_spec.build_regist_info_ODSS ( RMMC_1.GetId(), RW);
//register Audio_Packet_ODSS as an ODSS
Recv_SpM_spec.build_regist_info_ODSS ( Audio_Packet_ODSS.GetId(), RW);
// register Application-Workspace-Buffer as ODSS
Recv_SpM_spec.build_regist_info_ODSS ( m_SPK.GetId(), RW);
// register SpM
RegisterSpM ( (PFSpMBody) Recv_SpM, & Recv_SpM_spec);
}
```

Recv\_SpM registration

Feb-07

UCI  
DREAM Lab



## Receiver.cpp (con't)

```
int TMO_Recv_Class::Recv_SpM ()
```

◀BACK

```
{
```

```
    // Temporary variables used in SpM
```

```
    // Temporary variables used in SpM
```

```
    int      AudioMsgSize;
```

```
    int      result;
```

```
    MicroSec SpMStartTime = GetCurrentDCSage ();
```

```
    SAudioFrame AudioFrame;          //Play Buffer
```

```
    if (!m_SPK.IsSpkStart ()) return 0; // wait until speaker init is done
```

Feb-07

UCI  
DREAM Lab



## Receiver.cpp (con't)

```
while (1)
```

◀BACK

```
{    // Get frames from Audio_Packet_ODSS until returning  
    // there is no more frame there
```

```
    result = Audio_Packet_ODSS.CopyFrame (& AudioFrame);
```

```
    if (result != SUCCESS)
```

```
    {
```

```
        if (result == NO_VALUE)
```

```
        {
```

```
            _tprintf (_T("NO_VALUE\n"));
```

```
        }
```

```
        else if (result == FAIL)
```

```
            TMOSLprintf (_T("FAIL\n")); //TMOSLprintf ("FAIL\n");
```

```
            break;
```

```
    }
```

Feb-07

UCI  
DREAM Lab



## Receiver.cpp(con't)

```
else //Successfully get a frame from the ODSS
{
    TMOSLprintf (_T("TPT %dWn"), AudioFrame.aFrameID);
    m_SPK.Play ( (char*) AudioFrame.aData,
                WAVEOUT_AUDIO_BLOCK_SIZE);
}
}
return 1;
}
```

play an audio packet.

Feb-07

UCI  
DREAM Lab



## Receiver.cpp (con't)

```
TMO_Recv_Class::TMO_Recv_Class (TCHAR * TMO_name, tms & start_time):
    RMMC_1 (_T("RMMC_1"))
{
    //Initialize audio output device
    m_SPK.OpenSpk ();

    //Initialize SpM
    Recv_SpM_Register_Init ();

    Audio_Recv_SvM_Register_Init (); //Initialize SvM

    TMO_RegistParam TMO_Recv_spec;
    _tscopy (TMO_Recv_spec.global_name, TMO_name);
    TMO_Recv_spec.start_time = start_time;
    //Register TMO
    RegisterTMO (& TMO_Recv_spec);
}
```

Feb-07

UCI  
DREAM Lab



## Receiver.cpp (con't)

```
void TMO_Recv_Class::Audio_Recv_SvM_Register_Init ()
{
    SvM_RegistParam Audio_Recv_SvM_spec;
    // save SvM name
    _tscopy (Audio_Recv_SvM_spec.name, _T("Audio_Recv_SvM"));

    // register ODSS
    Audio_Recv_SvM_spec.build_regist_info_ODSS ( Audio_Packet_ODSS.GetId(), RW);

    // specify SvM GST
    Audio_Recv_SvM_spec.GETB = DEALINE_MSEC_RECEIVER_SVM * 1000;

    // register SvM
    if (RegisterSvM ((PFSvMBody) Audio_Recv_SvM, & Audio_Recv_SvM_spec) == FAIL)
        _tprintf (_T("Fail to register Audio_Recv_SvM object Wn"));
}
```

Feb-07

UCI  
DREAM Lab



## Receiver.cpp (con't)

```
void TMO_Recv_Class::Audio_Recv_SvM (SAudioFrame* param)
{
    // Save the audio packet into Audio_Packet_ODSS
    if (!Audio_Packet_ODSS.WriteAudioFrame (param))
        _tprintf (_T("Fail to write AudioFrame %d into ODSSWn"),
            param->aFrameID);
}
```

Feb-07

UCI  
DREAM Lab

