

EECS 223:
Real-Time Computer Systems
Lecture : RMMC

05.5.1	1
--------	---



Introduction

Basic Goals of Multicasts

- Two conceivable needs for group communications in RT distributed computing systems
 - [Server replicas](#) and [tightly coupled heterogeneous servers](#)
 - [News multicast](#) to casual readers
 - Attain high performance in operating tightly coupled servers and/or achieving casual news multicasts
- Two additional purposes
 - [Abstract distributed programming](#)
 - Basic building blocks for fault-tolerant distributed systems
 - Not important at this time because suitable / credible ways of doing this have not matured

05.5.1	2
--------	---



Introduction

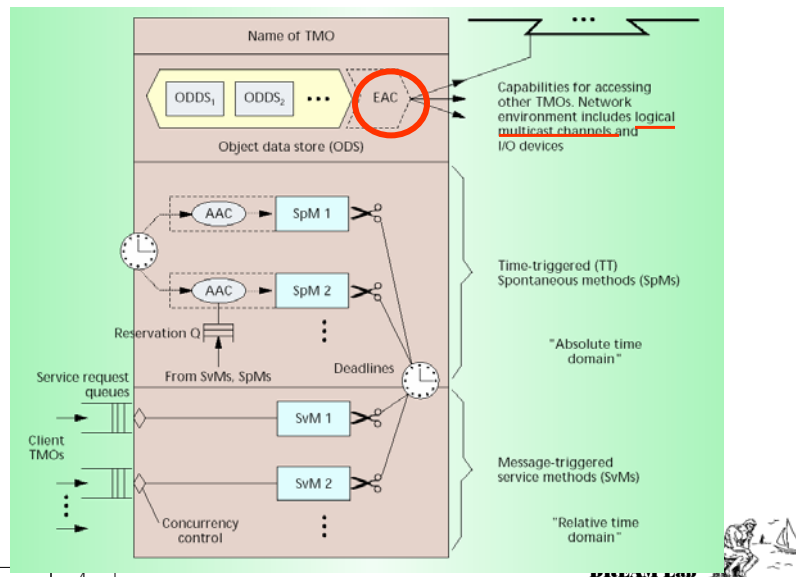
- Multicast channels for abstract distributed programming
 - TMOs can use another interaction mode in which messages can be exchanged over [logical message channels](#).
 - [Access gates](#) for such channels are explicitly specified as [data members](#), more specifically, [special types of ODSSs](#), of involved objects.
- [RMMC \(Real-time Multicast and Memory-replication Channel\)](#)
 - One of the most advanced types of multicast channels

05.5.1 | 3



Real-time Multicast and Memory-replication Channel (RMMC)

- Access gates for channels explicitly specified as data members of TMO objects

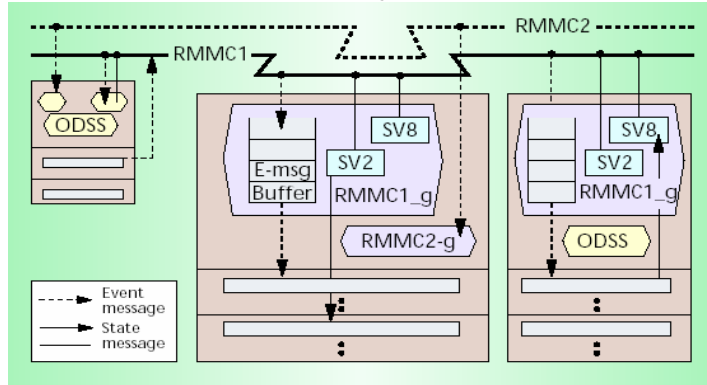


05.5.1 | 4



Real-time Multicast and Memory-replication Channel (RMMC)

RMMCs accessed by TMO methods



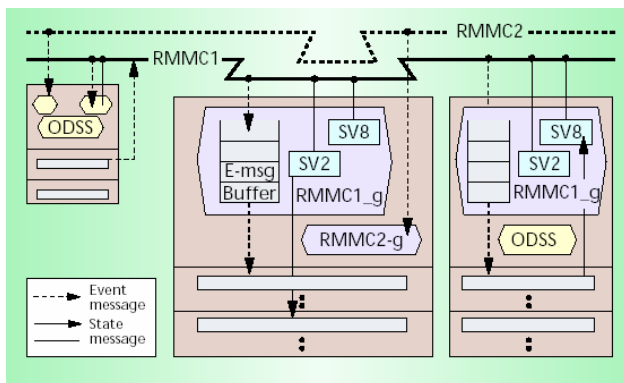
- Access gates for two RMMCs (RMMC1 and RMMC2) can be declared as **data members** (special types of ODSS's) of each of the 3 TMOs (which may be dispersed widely) during the design time.
- One gate is declared as **one subscriber** to an RMMC channel. One TMO can have **multiple gates** for the same RMMC channel.

05.5.1 5



RMMC

- An event message sent over RMMC1 will be delivered to the **buffer allocated inside the execution engine** for **each of the subscribing RMMC1-Gates** **except** the RMMC1-Gate through which the message goes out.
- Later, a certain method in each TMO with a receiving RMMC1-Gate can pick up those messages by sending requests **through the gate** to the exec engine.
- An RMMC can be implemented over point-to-point networks as well as over broadcast-enabled bus networks.



05.5.1 6



Event Messages and State Messages

- RMMC scheme supports not only conventional **event messages** but also **state messages** based on **distributed replicated memory semantics**.
- **Event message**
 - An event message **must be read by every corresponding subscriber** (RMMC-Gate) other than the announcing subscriber.
 - The announcing subscriber cannot obtain its event messages.
 - Every message sent must not be ignored by other subscribers.
 - If a subscriber doesn't read the message quickly enough, an **event message queue overflow** or a **time-out**, both of which are error detection events, can occur.

In principle, the programmer should have the option of specifying that a **message which has waited in the buffer for a maximum residency period** can be simply discarded without raising an error signal.
 - An event message producer **timestamps** the message at message-production time.

05.5.1	7
--------	---

Event Messages and State Messages

- **State message**
 - A state message carries information to be stored in **a fixed memory location in each subscriber** corresponding to the **ID of the state message**.
 - A state message's ID represents **a group of replicated memory units**, each capable of holding the information carried in the state message and belonging to a different subscriber.
 - A state message producer **timestamps** the message at message-production time.
 - A method in each TMO with a subscriber RMMC1-Gate can **read** the content of its state msg memory through the gate **at its convenient time**.
 - The producer may **update** the contents of the state message memory units **at a higher frequency** than the frequency at which a certain consumer reads the content of one of the state message memory units.
 - A state message is thus typically used to share the periodically observed state information about a dynamic state-varying item, e.g., a car's position.

05.5.1	8
--------	---

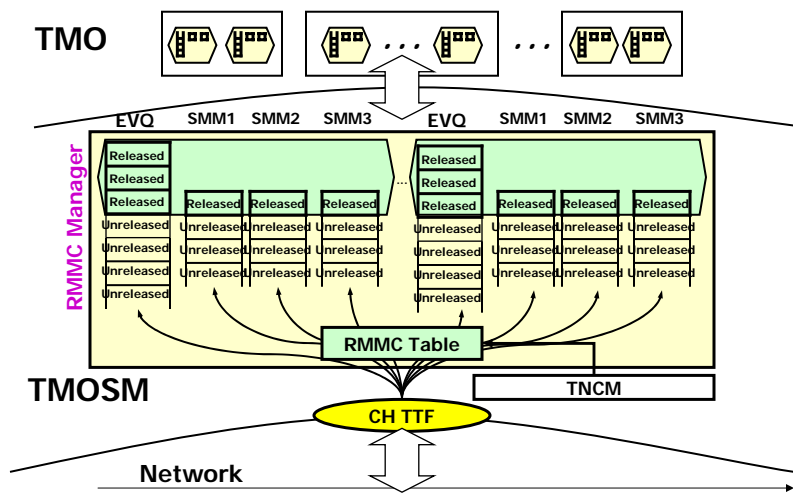
Official Release Time (ORT)

- The **official release time** (ORT) is the time at which the event / state message should become **accessible** through subscriber RMMC-Gates to the consumer methods.
- A (consumer) method in each TMO with a subscriber RMMC-Gate picks up **event messages** from the Q associated with the gate one at a time **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**.

05.5.1 9



RMMC and TMOSM



05.5.1 10



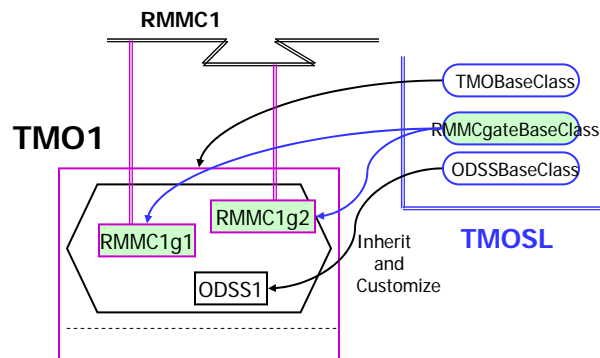
Official Release Time (ORT)

- **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 .

05.5.1 | 11



Creation of an RMMC Access Gate & an Enclosing TMO

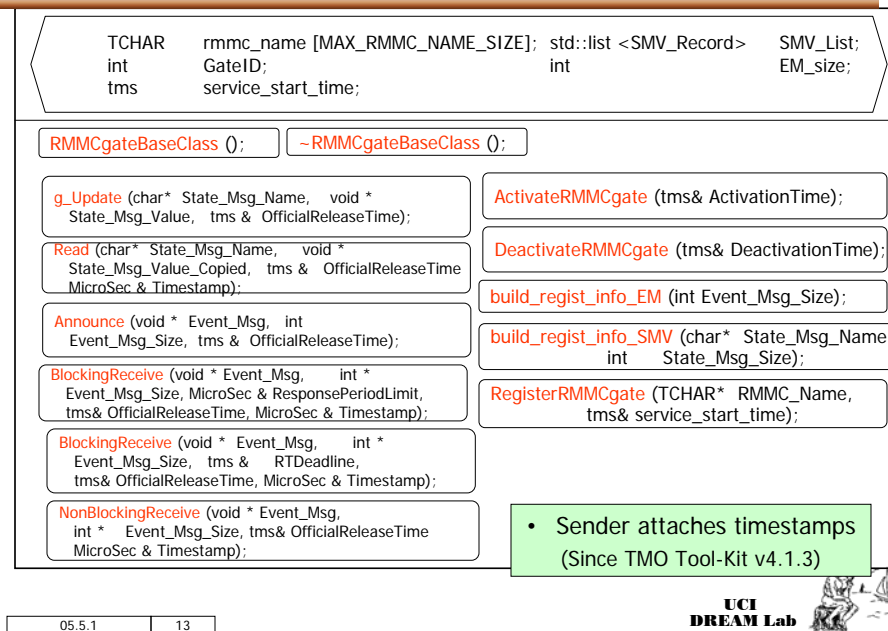


- The RMMC-Gate, not the TMO, is the unit for subscribing to an RMMC.
- Multiple RMMC-Gates for the RMMC may be used within a TMO.

05.5.1 | 12



RMMCgateBaseClass



API's

- Common APIs for establishing an RMMC-Gate

```

// Initialize the data members of RMMCgateBaseClass
- RMMCgateBaseClass::RMMCgateBaseClass ()

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

/* 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 RMMCgateBaseClass::build_regist_info_SMV (
    TCHAR * SM_name, int SM_size);

```

05.5.1

14

API's

- Common APIs for establishing a RMMC

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

- **void RMMCgateBaseClass::build_regist_info_EM** (int EM_size)

```
// Create one support record inside TMOSM
// for a gate to the specified RMMC.
```

- **BOOL RMMCgateBaseClass::RegisterRMMCgate** (
TCHAR* RMMC_Name, tms & service_start_time)

05.5.1	15
--------	----



API's (cont.)

- Event Message API's

```
/* Announce an "event_msg" to all subscribers (RMMC-Gates) of the
subject RMMC except for the announcing subscriber.
```

Each of the TMOSM nodes hosting subscribers will create a copy of this message for each local subscriber as the message comes in.

The "OfficialReleaseTime" indicates when this event message should be released to each subscriber.

If "OfficialReleaseTime" is not provided or if it is set to "0", this event message will be released to subscribers ASAP.

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

- int **RMMCgateBaseClass::Announce** (void* event_msg,
int Msg_size, tms OfficialReleaseTime)

05.5.1	16
--------	----



API'S (cont.)

- Event Message API's

/* Block until an "event_msg" is received or the amount of time indicated by "ResponsePeriodLimit" or "RTDeadline" passes by.

Returns SUCCESS, FAIL (when failure signals have been raised by TMOSM), or TIMEOUT indicating inability to complete the message pickup operation within ResponsePeriodLimit while no other failure signals have been raised by TMOSM.

After completion of this function, the result parameter "OfficialReleaseTime" contains the ORT that was attached to "event_msg". In case "OfficialReleaseTime" is 0, this event message is delivered to subscribers ASAP.

The result parameter "Timestamp" contains the "send-timestamp" created by the announcing subscriber.*/

- int **RMMCgateBaseClass::BlockingReceive** (void* event_msg, int * Received_Msg_size, microsec & ResponsePeriodLimit, tms & OfficialReleaseTime, MicroSec & Timestamp);
- int **RMMCgateBaseClass::BlockingReceive** (void* event_msg, int * Received_Msg_size, tms & RTDeadline, tms & OfficialReleaseTime, MicroSec & Timestamp);

05.5.1

17

API'S (cont.)

- Event Message API's

/* Checks if an "event_msg" has arrived and if it has, picks it up.

Returns SUCCESS, FAIL (when failure signals have been raised by TMOSM), or NO_VALUE without blocking.

After completion of this function, the result parameter "OfficialReleaseTime" contains the ORT that was attached to "event_msg".

In case "OfficialReleaseTime" is 0, this event message is delivered to subscribers ASAP.

The result parameter "Timestamp" contains the "send-timestamp" created by the announcing subscriber. */

- int **RMMCgateBaseClass::NonBlockingReceive** (void* event_msg, int * Received_Msg_size, tms & OfficialReleaseTime, MicroSec & Timestamp)

05.5.1

18

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 RMMCgateBaseClass::g_Update (char* StateMsg_Name,
  void * StateMsg_Value_To_Be_Stored,
  tms OfficialReleaseTime)
```

05.5.1

19

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

05.5.1

20

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 RMMCgateBaseClass::DeactivateRMMCgate (
    tms & Deactivation_Time)

```

05.5.1	21
--------	----



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 RMMCgateBaseClass::ActivateRMMCgate (
    tms & Reactivation_Time)

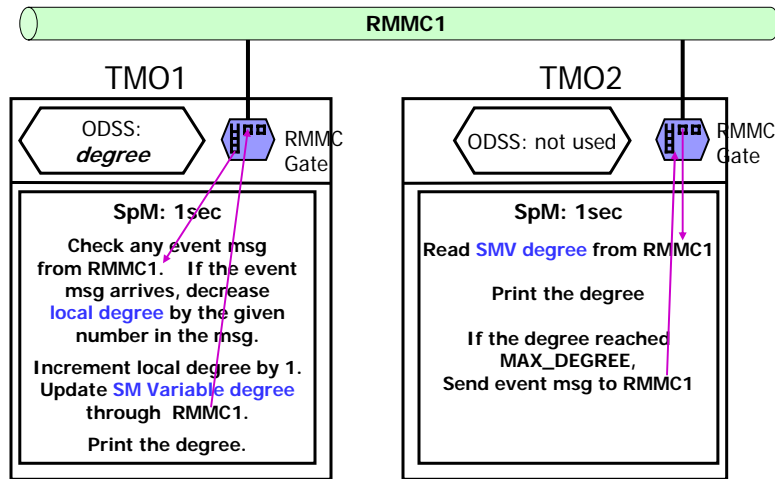
```

05.5.1	22
--------	----



Example

- This example consists of two TMOs which share one RMMC



05.5.1 23

UCI
DREAM Lab

Example (cont.)

- RMMC.h

```
#define MAX_DEGREE 5

// Struct for State Msgs
struct SM1struct_RMMC1
{
    int degree;
};

// Struct for Event Msg (also can be defined dynamically)
struct EMstruct_RMMC1
{
    int degree;
    TCHAR Message[30];
};
```

05.5.1 24

UCI
DREAM Lab

Example (cont.)

- RMMC.h (cont.)

```
// RMMC for state message Comm
class CRMMCGateClass: public RMMCgateBaseClass
{
public:
    CRMMCGateClass (TCHAR* RMMCname)
    {
        // build state message info
        build_regist_info_SMV (_T("SM1"), sizeof (SM1struct_RMMC1));
        // build event message info
        build_regist_info_EM (sizeof (EMstruct_RMMC1));
        // RMMC gate Registration
        RegisterRMMCgate (RMMCname);
    };
};
```

05.5.1

25

Example (cont.)

- TMO1.h

```
#include "RMMC.h"
class ODSSClass1: public ODSSBaseClass <ODSSClass1> {
public:
    int Degree;
    ODSSClass1 () { Degree = 0; }
};

class TMO1: public CTMOBase
{
public:
    TMO1 (TCHAR *, TCHAR*, tms);
private:
    ODSSClass1          ODSS1;
    int                 SpM1 ();
    CRMMCGateClass     RMMC_TMO1;
};
```

05.5.1

26

Example (cont.)

- TMO1.cpp

```
TMO1::TMO1 (TCHAR * TMO_name, TCHAR* sRMMCName,
           tms TMO_start_time)
           :RMMC_TMO1 (sRMMCName)
{
    // register SpM
    AAC aac1 (NULL,           // null for candidate aac label
             tm4_DCS_age (1*1000*1000), // from
             tm4_DCS_age (60*1000*1000), // until
             1 * 1000 * 1000, // every
             0,                // EST
             100*1000,         // LST
             200*1000         // by
            );
}
```

05.5.1

27



Example (cont.)

- TMO1.cpp

```
SpM_RegistParam spm_spec;
spm_spec.build_regist_info_AAC (aac1);
spm_spec.build_regist_info_ODSS (ODSS1.GetId(), RW);
spm_spec.build_regist_info_ODSS (RMMC_TMO1.GetId(), RW);
RegisterSpM ( (PFSpMBody) SpM1, & spm_spec);

// register TMO
TMO_RegistParam tmo_spec;
_tcscopy (tmo_spec.global_name, TMO_name);
tmo_spec.start_time = TMO_start_time;
RegisterTMO (& tmo_spec);
}
```

05.5.1

28



Example (cont.)

- TMO1.cpp (cont.)

```
int TMO1::SpM1 ()
{
    // Variables for RMMC
    int          degree;
    tms          OfficialReleaseTime;
    int          sizeofEM;
    MicroSec     Timestamp;
    static SM1struct_RMMC1      SM1_RMMC1;
    static EMstruct_RMMC1      EMPara;

    // Start SpM
    degree = ODSS1.Degree;
```

05.5.1

29

UCI
DREAM Lab

Example (cont.)

- TMO1.cpp (cont.)

```
// RMMC check any event message
if ( RMMC_TMO1.NonBlockingReceive (&EMPara, &sizeofEM,
    OfficialReleaseTime, Timestamp) == SUCCESS){
    degree -= EMPara.degree;
    TMOSLprintf (_T("Message from TMO2:"));
    TMOSLprintf (EMPara.Message);
}
SM1_RMMC1.degree = degree;
// Update State Message in RMMC1
OfficialReleaseTime = later (tm4_DCS_age (GetCurrentDCSage ()), 10000);
// This message is valid, starting in 10 millisecs from now.
RMMC_TMO1.g_Update (_T("SM1"), &SM1_RMMC1,
    sizeof(SM1struct_RMMC1), OfficialReleaseTime);
TMOSLprintf (_T("1.Current Degree: %d\n"),degree);
// Increment Degree
ODSS1.Degree = ++degree;
// End SpM
return 1;
```

05.5.1

30

UCI
DREAM Lab

Example (cont.)

- TMO2.h

```
class TMO2: public CTMOBase
{
public:
    TMO2 (TCHAR *, TCHAR *, tms);
private:
    ODSSClass1          ODSS1;
    int                SpM2 ();
    CRMMCClass         RMMC_TMO2;
};
```

05.5.1

31



Example (cont.)

- TMO2.cpp

```
TMO2::TMO2 (TCHAR * TMO_name, TCHAR * sRMMCName,
            tms TMO_start_time)
:RMMC_TMO2 (sRMMCName)
{
    // register SpM
    AAC aac1 (NULL, // null for candidate aac label
             tm4_DCS_age (1*1000*1000), // from
             tm4_DCS_age (60*1000*1000), // until
             1 * 1000 * 1000, // every
             0, // EST
             100*1000, // LST
             200*1000 // by
    );
```

05.5.1

32



Example (cont.)

- TMO2.cpp

```
SpM_RegistParam spm_spec;
spm_spec.build_regist_info_AAC (aac1);
spm_spec.build_regist_info_ODSS (ODSS1.GetId (), RW);
spm_spec.build_regist_info_ODSS (RMMC_TMO2.GetId (), RW);
RegisterSpM ( (PFSpMBody) SpM1, & spm_spec);
```

05.5.1

33



Example (cont.)

- TMO2.cpp (cont.)

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

int TMO2::SpM2 ()
{
    // Variables for RMMC
    MicroSec      Timestamp;
    tms           OfficialReleaseTime;
    static SM1struct_RMMC1      SM1_RMMC1;
    static EMstruct_RMMC1      EMPara;
    // SpM started
```

05.5.1

34



Example (cont.)

- TMO2.cpp (cont.)

```
// RMMC
if ( RMMC_TMO2.Read (_T("SM1"), &SM1_RMMC1, OfficialReleaseTime, Timestamp)
    == SUCCESS)
{
    TMOSLprintf (_T(" 2.Current Degree: %d\n"), SM1_RMMC1.degree);
    if ( SM1_RMMC1.degree >= MAX_DEGREE )
    {
        // Send TMO1 Event Message to decrease the degree
        EMPara.degree = MAX_DEGREE;
        _stprintf (EMPara.Message, _T("Please reduce the degree by %d\n"),
            EMPara.degree);
        OfficialReleaseTime = later (tm4_DCS_age (GetCurrentDCSage ()),15000);
        // This message is valid, starting in 15 millisecs from now.
        RMMC_TMO2.Announce (&EMPara, sizeof (EMPara), OfficialReleaseTime );
    }
};
}
return 1;
}
```

05.5.1

35



Example (cont.)

- TestTMO.cpp

```
int _tmain (int argc, _TCHAR* argv[])
{
    // Start TMO support Middleware
    StartTMOengine ();

    tms TMO_start_time1 = tm4_DCS_age (2*1000*1000);
    TMO1 T1 (_T("TMO1"), _T("RMMC1"), TMO_start_time1);

    tms TMO_start_time2 = tm4_DCS_age (2*1000*1000);
    TMO2 T2 (_T("TMO2"), _T("RMMC1"), TMO_start_time2);

    MainThrSleep ();

    return 0;
}
```

05.5.1

36



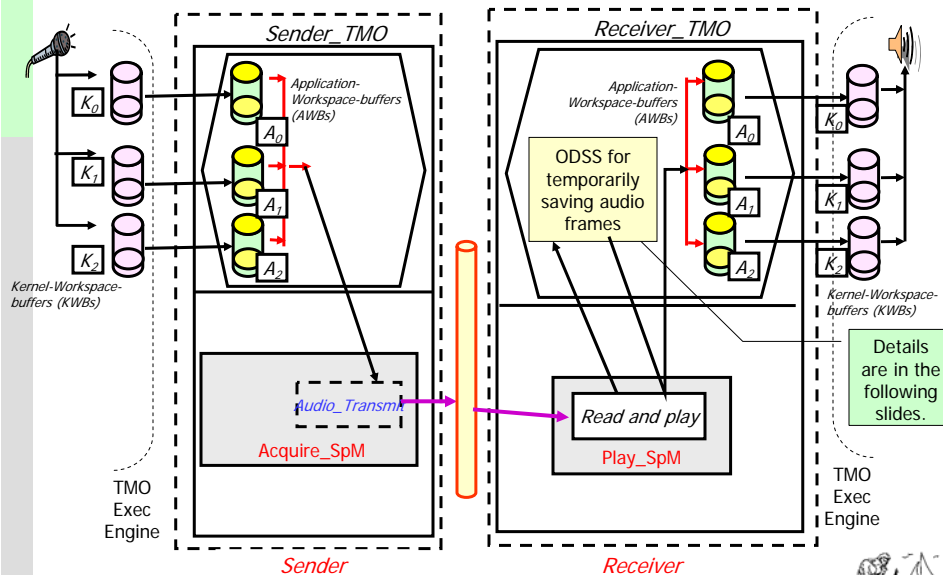
Tele-Audio Application

- 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.
- Receiver has one SpM which receives data packets from the sender and writes the received voice data into audio device driver.
- There is one RMMC channel between sender and receiver which facilitates message exchanges.

05.5.1 37



TMO Network for Tele-Audio Application

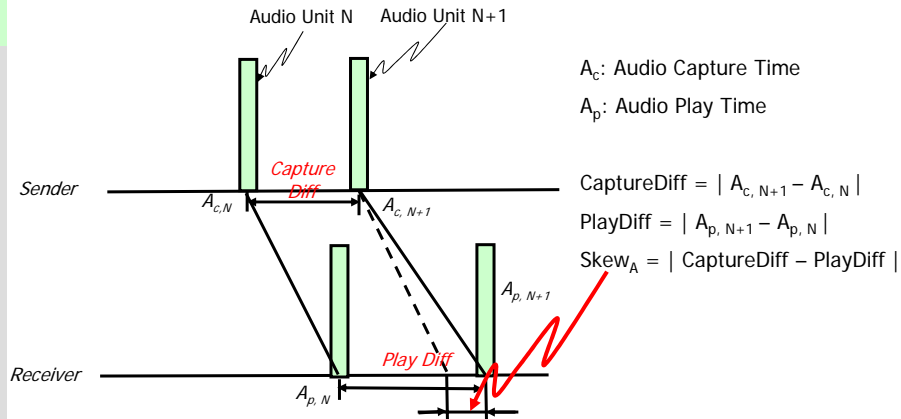


05.5.1 38



Intra-stream Synchronization

- Refers to the synchronization between Media Units (MUs) within one media stream, such as audio or video. The temporal relationship between MUs of the same stream should be kept consistency at the sending and receiving sides.



Goal: For audio stream, the jitter of the play times of two successive media units in one stream should be as small as possible.

05.5.1

39

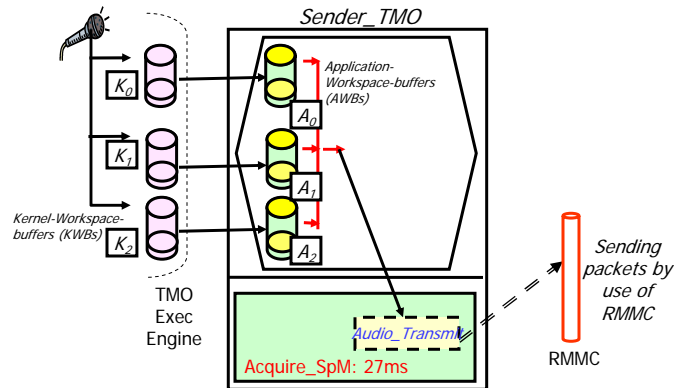
Imaginary Capture Time (ICT) and Target Play Time (TPT) Scheme

- An effective approach is used for [intra-stream synchronization](#).
- If every media unit can be played at its TPT, and also the difference of the TPTs between two successive media units is kept constant, intra-stream synchronization can be readily achieved.
- TPT of one MU is determined by adding a fixed delay to its capture time. The fixed delay added is called the TSD ([Target Streaming Delay](#))
- Because TPT is impacted by the capture time, a technique for realizing [high-precision capture-time-stamping](#) is a key.
 - Since there is no way of creating capture-timestamps in 'perfect real time', an [Imaginary Capture-Time-stamp \(ICT\)](#) is attached to each media unit.
 - TPT = ICT + TSD

05.5.1

40

Design of Sender



- In the initialization phase of audio capturing, three application-workspace-buffers (AWBs) are registered into the OS kernel.
- For each AWB, a corresponding kernel-workspace-buffer (KWB) with the same size is allocated in the kernel.
- When the capturing starts, audio device driver saves audio packets into KWBs sequentially, i.e., $K_0, K_1, K_2, K_0, K_1, K_2, K_0, \dots$
- To obtain an audio packet from a KWB, its corresponding AWB should be provided.
- In order to obtain audio packets sequentially, the sequence of providing AWBs is $A_0, A_1, A_2, A_0, \dots$

05.5.1 41

UCI DREAM Lab

Wrapper Class for Audio Input Device

Mic_Wrapper_Class (Inherited from ODSSBaseClass)

```

char    AWB [IN_BUFFER] [WAVEIN_AUDIO_BLOCK_SIZE];
        // Application workspace buffer
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.
    
```

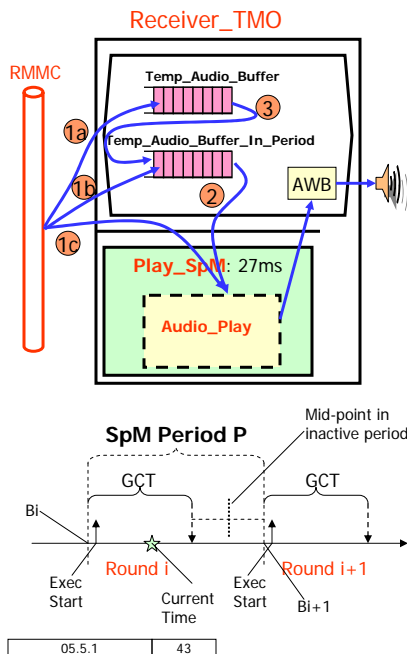
```

Mic_Wrapper_Class (); // constructor
void OpenMic (); // Open the audio input device.
BOOL IsMicStart (); // Is it ready to use?
MMRESULT Read (int buffer_number); // Read audio data from device driver.
Char * GetBufferData (int buffer_number); // Get a pointer to AWB.
    
```

05.5.1 42

UCI DREAM Lab

Design of Receiver



- Step 1: When a packet is received from RMMC, it will be treated differently based on its TPT.
 - Case 1a: $TPT < \text{Current Time}$
 - Audio packet should be played immediately.
 - Case 1b: $\text{Current Time} < TPT < B_i + GCT + \frac{1}{2} * (P-GCT)$
 - Audio packet is saved into Temp_Audio_Buffer_In_Period and waits until TPT arrives to be played.
 - Case 1c: $TPT > B_i + GCT + \frac{1}{2} * (P-GCT)$
 - Audio packet is saved into Temp_Audio_Buffer and waits until the next round of Play_SpM.
- Step 2: An audio packet is retrieved from Temp_Audio_Buffer_In_Period. If the TPT of this audio packet is close to the current time (within a small time window), this audio packet is played.
- Step 3: An audio packet is retrieved from Temp_Audio_Buffer. If $TPT < B_{i+1} + GCT + \frac{1}{2} * (P-GCT)$, this audio packet is moved into Temp_Audio_Buffer_In_Period and waits until TPT arrives to be played.

Wrapper Class for Audio Output Device

Spk_Wrapper_Class (Inherited from ODSSBaseClass)

```

char AWB [OUT_BUFFER] [WAVEIN_AUDIO_BLOCK_SIZE];
    // Application workspace buffer
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.

Spk_Wrapper_Class (); // constructor
void OpenSpk (); // Open the audio output device.
BOOL IsSpkStart (); // Is it ready to use?
void Play (char * waveout_dataPtr, int waveout_size);
    // Method for Playing Audio Frames
    
```

Visual Studio Project (VSP) Structures

TeleAudioMono_Sender

Source Files

- [main.cpp](#) — Initialize TMOSM and create Sender TMO
- [Sender.cpp](#) — Major Source File (explained in the following slides)
- [Mic_Wrapper.cpp](#) — Wrapper class for audio-in device
- StdAfx.cpp — “empty”

Header Files

- [constants.h](#) — Definitions of constants
- [PacketFormat.h](#) — Definition of Audio Frame and A class is used to instantiate application_workspace_buffers
- [RMMC.h](#) — Definition of Application-defined RMMC-Gate Class
- [StdAfx.h](#) — Declarations of Common Included Header Files, such as tmosl.h
- [Sender.h](#) — Major Header File (Explained in the following slides)
- [Mic_Wrapper.h](#) — Wrapper class header for audio-in device

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

05.5.1 45

UCI
DREAM Lab



Visual Studio Project (VSP) Structures

TeleAudioMono_Receiver

Source Files

- [main.cpp](#) — Initialize TMOSM and create Receiver TMO
- [Receiver.cpp](#) — Major Source File (Explained in the following slides)
- [Spk_Wrapper.cpp](#) — Wrapper for audio-out device
- StdAfx.cpp
- [CircularBuffer.cpp](#) — Source File of Circular Buffer

Header Files

- constants.h
- PacketFormat.h
- RMMC.h
- StdAfx.h
- [Receiver.h](#) — Major Header File (Explained in the following slides)
- [CircularBuffer.h](#) — Header File of Circular Buffer
- [Spk_Wrapper.h](#) — Header File of audio-out device wrapper

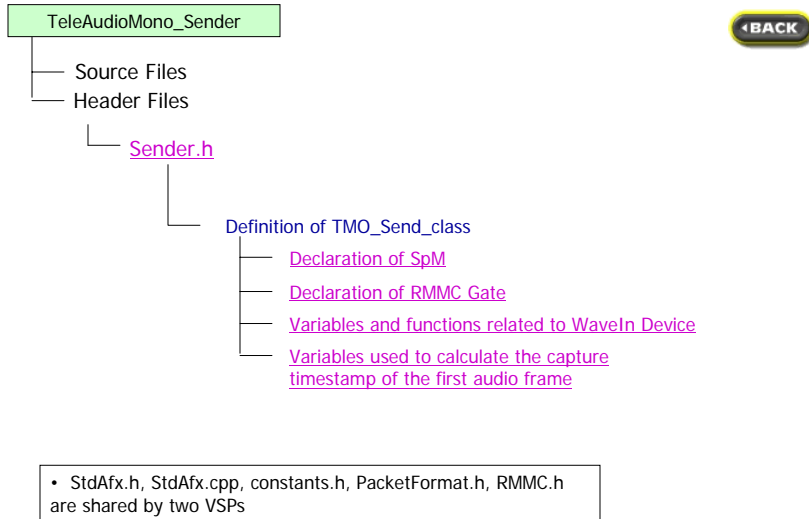
- StdAfx.h, StdAfx.cpp, constants.h, PacketFormat.h, RMMC.h are shared by two VSPs

05.5.1 46

UCI
DREAM Lab



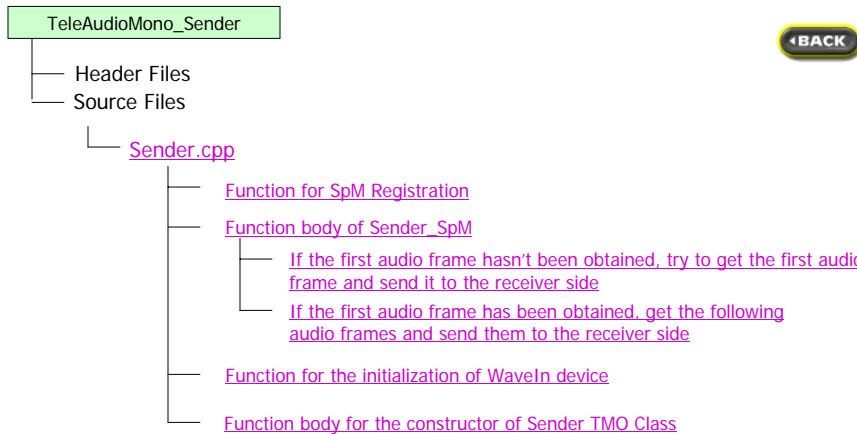
TeleAudioMono_Sender::Sender.h



05.5.1 47



TeleAudioMono_Sender::Sender.cpp



05.5.1 48



TeleAudioMono_Receiver::Receiver.h

TeleAudioMono_Receiver

◀BACK

Source Files
Header Files

Receiver.h

Definition of TMO_Recv_class

Declaration of SpM

Declaration of RMMC Gate

Two circular buffers to save frames arriving earlier than their TPTs

Variables and functions related to WaveOut Device

05.5.1 49

UCI
DREAM Lab



TeleAudioMono_Receiver::Receiver.cpp

TeleAudioMono_Receiver

◀BACK

Header Files
Source Files

Receiver.cpp

Function for playing audio frames

Function for the initialization of Waveout device

Function for the registration of SpM of Receiver TMO Class

Function body of Recv_SpM

Check Temp_Audio_Buffer to decide that an audio frame should be played immediately, moved into Temp_Audio_Buffer_In_Period, or still kept in Temp_Audio_Buffer based on its TPT.

Check RMMC channel to receive audio frames. Based on the TPT of an audio frame, it is played immediately, moved into Temp_Audio_Buffer_In_Period, or moved into Temp_Audio_Buffer.

Check Temp_Audio_Buffer_In_Period to play an audio frame at its TPT

Function body for the constructor of Receiver TMO Class

05.5.1 50

UCI
DREAM Lab



constants.h()

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

// SpM-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
```

05.5.1

51

UCI
DREAM Lab



constants.h() (cont')

```
#define SAMPLE_RATE 8000
#define SAMPLE_SIZE 1
#define CHANNEL_NUM 1
#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 148 // Audio Target Streaming Delay
#define MAX_PLAY_IN_ONE_PERIOD 10
#define AUDIO_CIRCLE_LENGTH 21
#define AUDIO_DIRECT_PLAY_RANGE 3
```

// WaveIn & WaveOut device-related constants

05.5.1

52

UCI
DREAM Lab



PacketFormat.h

```


#ifndef __PACKET_FORMAT_H__
#define __PACKET_FORMAT_H__

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

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

```

// The data structure of a frame

**UCI
DREAM Lab** 

05.5.1

53

RMMC.h


```

#ifndef RMMC_H
#define RMMC_H
#include "stdafx.h"
#include "PacketFormat.h"

class RMMCGateClass: public RMMCgateBaseClass // RMMCGateClass is inherited
                                                         from RMMCgateBaseClass
{
public:
    RMMCGateClass (TCHAR* RMMC_name)
    {
        // build event message info
        build_regist_info_EM (sizeof (SAudioFrame));

        // RMMC gate Registration
        RegisterRMMCGate (RMMC_name);
    };
};
#endif // RMMC_H

```

**UCI
DREAM Lab** 

05.5.1

54

stdAfx.h

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



// 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
```

05.5.1

55

UCI
DREAM Lab



stdAfx.h (cont')

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

```
//#define AUDIO_RMMC_to_SvM
#include <tchar.h>
#include <windows.h>
#ifdef _WIN32_WCE
#include <vfw.h>
#else
#include <Mmsystem.h>
#endif
```

```
// TMOsl header
```

```
#include <tmosl.h>
#ifdef _WIN32_WCE
#pragma comment ( lib, "winmm")
#endif
#endif
```

05.5.1

56

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 () {};

```



05.5.1

57




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

```

05.5.1

58




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;
}
```

05.5.1

59





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;
}
```

05.5.1

60




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;
    TMOSLprintf (_T("*** WaveIn device open failed.***\n"));
    return ;
}

```

05.5.1

61

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);

```

05.5.1

62

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;
}

```

05.5.1	63
--------	----

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

```

05.5.1	64
--------	----

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 ();
};

```

05.5.1	65
--------	----

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

05.5.1	66
--------	----

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

05.5.1

67

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.

05.5.1

68

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.

05.5.1

69

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

05.5.1

70

UCI
DREAM Lab

CircularBuffer.cpp (con't)

```

unsigned int Temp_Audio_Class::GetAudioBufferHead () {
    return AudioBufferHead;
}
unsigned int Temp_Audio_Class::GetAudioBufferTail () {
    return AudioBufferTail;
}
// The function
// bodies of auxiliary
// methods
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;
}

```

UCI
DREAM Lab



05.5.1

71

Sender.h

```

#ifndef __SENDER_H__
#define __SENDER_H__

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

using namespace TMO;

```

◀BACK

UCI
DREAM Lab



05.5.1

72

Sender.h (con't)

```
// class : TMO_Class
class TMO_Send_Class: public CTMOBase
{
private:
    int  Sender_SpM (); // SpM body
    void Sender_SpM_Register_Init (); // SpM Initialization Method
    RMMCGateClass    RMMC_1;        // RMMC gate

    Mic_Wrapper_Class    m_MIC; // wrapper object for audio-in device
```

◀BACK

05.5.1

73

UCI
DREAM Lab



Sender.h (con't)

__int64	A_nMessageID;	//Frame counter
__int64	SpM_iteration;	// Counter for SpM iteration
MicroSec	Audio_Start_Time;	

unsigned char	LastBufferCount;	// Auxiliary variables
bool	bGetFirstAudioFrame;	
bool	BufferFull [IN_BUFFER];	

// Variables used to calculate the capture timestamp of the first audio frame

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

◀BACK

05.5.1

74

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 TMOSM takes the control
    MainThrSleep ();
}

```

◀BACK

An alternative way to instantiate an TMO obj:

```

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

```

05.5.1	75
--------	----

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 ()
{
    int i;
    bGetFirstAudioFrame = false;
    LastBufferCount = 0;
    for(i = 0; i < IN_BUFFER ; i++) {
        BufferFull[i] = false;
    }
}

```

◀BACK

05.5.1	76
--------	----

UCI
DREAM Lab



Sender.cpp (con't)

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

◀BACK

//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

05.5.1

77

UCI
DREAM Lab



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);
// Register RMMC gate as an ODSS
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
if (RegisterSpM ((PFSpMBody) Sender_SpM, &Sender_SpM_spec) == FAIL)
    TMOSLprintf (_T("Fail to register Sender_SpM object \n"));
}
```

// Sender SpM Registration

05.5.1

78

UCI
DREAM Lab



Sender.cpp (con't)

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

```
{
```

```
    SAudioFrame      AudioFrame;
    MicroSec SpM_starttime = GetCurrentDCSage ();
    MicroSec CheckTime;

    HRESULT hResult;

    MMRESULT      result;
    WAVEHDR      *pWaveHdr;
    MMTIME      typetime;

    unsigned char cTemp = 0;
    int i, j;
```

// Temporary
variables used in
SpM

◀BACK

05.5.1

79

UCI
DREAM Lab



Sender.cpp (con't)

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

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

```
{
```

```
    //clean up audio buffers at the first checking of Sender SpM
```

```
    if (SpM_iteration == 0)
```

```
    {
```

```
        // For every input buffer.
```

```
        for (i = 0; i < IN_BUFFER ; i++)
```

```
        {
```

```
            m_MIC.Read(i);
```

```
        }
```

```
    }
```

◀BACK

05.5.1

80

UCI
DREAM Lab



Sender.cpp (con't)

```
else // If these is not the first checking
```

```
{
```

```
    if(!bGetFirstAudioFrame) // If the buffer that the first audio resides is unknown
    {
        CheckTime = GetCurrentDCSage();
        // Clean up all audio buffers
        for (i = 0; i < IN_BUFFER ; i++)
        {
            BufferFull[i] = false;
            result = m_MIC.Read(i);
            if(result == MMSYSERR_NOERROR)
            {
                BufferFull [i] = true;
                cTemp++;
            }
        }
    }
}
```

Try to get
the first
audio frame
from
buffers
(con't)

05.5.1

81

UCI
DREAM Lab



Sender.cpp (con't)

```
if(cTemp > 0)
```

```
{
```

```
    //all buffers are full again. Needs to clean up again.
```

```
    if(cTemp == IN_BUFFER)
```

```
    {
```

```
        for (i = 0; i < IN_BUFFER ; i++)
```

```
        {
```

```
            result = m_MIC.Read(i);
```

```
        }
```

```
    }
```

Try to get the first audio frame from buffers (con't)

05.5.1

82

UCI
DREAM Lab



Sender.cpp (con't)

else //If not all buffers are buffer, locate the position of the buffer which contains the first frame

```
{
    // Try to get the first audio frame from buffers
    for (i = 0; i < IN_BUFFER ; i++)
    {
        if (BufferFull[i] == false) break;
    }
    bool bGetFirstBuffer = false;
    for (j = i; j < IN_BUFFER ; j++)
    {
        if (BufferFull[j] == true)
        {
            bGetFirstBuffer = true;
            break;
        }
    }
}
```

05.5.1

83

UCI
DREAM Lab

Sender.cpp (con't)

```
if (!bGetFirstBuffer) {
    j = 0;
}
for (i = 0; i < cTemp; i++) {
    if (i == 0) { // First data
        AudioFrame.aICT = CheckTime - cTemp * SENDER_SPM_PERIOD * 1000 -
            SENDER_SPM_PERIOD * 1000 / 2;
        Audio_Start_Time = AudioFrame.aICT;
    }
    else AudioFrame.aICT = Audio_Start_Time + A_nMessageID * SENDER_SPM_PERIOD
        * 1000;
    AudioFrame.aFrameID = A_nMessageID;
    AudioFrame.aTPT = AudioFrame.aICT + AUDIO_TSD * 1000;
    memcpy (AudioFrame.aData, m_MIC.GetBufferData(j),
        WAVEOUT_AUDIO_BLOCK_SIZE);
}
```

// If the first audio frame is obtained, calculate the capture timestamp of the first frame

05.5.1

84

UCI
DREAM Lab

Sender.cpp (con't)

```

if (j + 1 == IN_BUFFER) j = j + 1 - IN_BUFFER;
else j++;

AudioFrame.aSend = GetCurrentDCSage();

if (!IRMMC_1.Announce ( (void *) &AudioFrame, sizeof (SAudioFrame)) == SUCCESS)
    TMOSLprintf (_T("Fail to send Audio Signal %d Successfully\n"),
        AudioFrame.aFrameID);
else TMOSLprintf (_T("Success to Send Audio Signal %d\n"), AudioFrame.aFrameID);

A_nMessageID++;
}
LastBufferCount = j;
bGetFirstAudioFrame = true;
TMOSLprintf (_T("GetFirstFrame\n")); // Indicate the first frame is acquired
}
}
}

```

// Send out the first frame

05.5.1

85

UCI
DREAM Lab



Sender.cpp (con't)

else // The procedure that other frames except for the first frame is shown in the following three slides

```

{
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);
    }
}
}

```

←BACK

// Get an audio frame other than the first audio frame from a buffer

05.5.1

86

UCI
DREAM Lab



Sender.cpp (con't)

```

if (result == MMSYSERR_NOERROR) {
    // Prepare audio frame to be sent out
    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.aICT = Audio_Start_Time + A_nMessageID * SENDER_SPM_PERIOD *
        1000;
    AudioFrame.aFrameID = A_nMessageID;
    AudioFrame.aTPT = AudioFrame.aICT + AUDIO_TSD * 1000;
    TMOSLprintf(_T("From Buffer %d for %d\n"), LastBufferCount, A_nMessageID);
    AudioFrame.aSend = GetCurrentDCSage ();
}

```

05.5.1

87

Sender.cpp (con't)

```

if (!IRMMC_1.Announce ((void *)&AudioFrame, sizeof(SAudioFrame)) == SUCCESS)
    TMOSLprintf (_T("Fail to send Audio Signal %d Successfully\n"),
        AudioFrame.aFrameID);
else
    TMOSLprintf (_T("Success to Send Audio Signal %d\n"),
        AudioFrame.aFrameID);
    A_nMessageID++;
}
else
    break;
}
}
}
}
}
SpM_iteration++;

return 1;
}

```

// Send out the frames

05.5.1

88

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;
    _tcscopy (TMO_Send_spec.global_name, TMO_name);
    TMO_Send_spec.start_time = start_time;
    RegisterTMO (&TMO_Send_spec);
}

```



05.5.1

89




Receiver.h

```

#ifndef __RECEIVER_H__
#define __RECEIVER_H__

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

using namespace TMO;

```



05.5.1

90




Receiver.h (con't)

```
// class : TMO_Class
class TMO_Recv_Class: public CTMOBase
{
private:
    int          Recv_SpM (); // SpM method
    RMMCGateClass RMMC_1;    // RMMC gate
    void         Recv_SpM_Register_Init (); // SpM initialization method

    Spk_Wrapper_Class m_SPK; // wrapper object for audio-out device
```

◀BACK

05.5.1

91

UCI
DREAM Lab



Receiver.h (con't)

```
__int64      SpM_iteration; // Counter for Recv_SpM iterations
unsigned int  A_nMessageID;
unsigned char CircleLength; // Auxiliary variables
```

```
Temp_Audio_Class TempAudioBufferInPeriod;
Temp_Audio_Class Temp_Audio_Buffer; // Two circular buffers to save frames arriving earlier than their TPTs
```

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

◀BACK

05.5.1

92

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 () {};

```



05.5.1	93
--------	----

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

```

05.5.1	94
--------	----

UCI
DREAM Lab




Spk_Wrapper.cpp

```
#include "Spk_Wrapper.h"

// Function for playing audio frames
// Play out audio frames
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));
        }
    }
}

```

UCI DREAM Lab 


05.5.1

95

Spk_Wrapper.cpp (con't)

```
switch (res) {
    case MMSYSERR_INVALIDHANDLE:
        TMOSLprintf(_T("waveOutWrite MMSYSERR_INVALIDHANDLE\n"));
        break;
    case MMSYSERR_NODRIVER:
        TMOSLprintf(_T("waveOutWrite MMSYSERR_NODRIVER\n"));
        break;
    case MMSYSERR_NOMEM:
        TMOSLprintf(_T("waveOutWrite MMSYSERR_NOMEM\n"));
        break;
    case WAVERR_UNPREPARED:
        TMOSLprintf(_T("waveOutWrite WAVERR_UNPREPARED\n"));
        break;
    case MMSYSERR_NOERROR:
        break;
}
} // if
} // for
} // All kinds of error messages when
// failing to play out audio frames

```

UCI DREAM Lab 

05.5.1

96

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

05.5.1

97

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 );

```

05.5.1

98

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.\n"));
    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;
}

```

05.5.1

99

UCI
DREAM Lab

main() (Receiver)

```

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

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 TMOSM takes the control
    MainThrSleep ();
}

```

05.5.1

100

UCI
DREAM Lab

Receiver.cpp

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

◀BACK

05.5.1

101

UCI
DREAM Lab 

Receiver.cpp (con't)

```
// Initialize SpM
void TMO_Recv_Class::Recv_SpM_Register_Init ()
{
    SAudioFrame          AudioFrame;
    CircleLength = AUDIO_CIRCLE_LENGTH - 3;
    SpM_iteration = 0;

    SpM_RegistParam Recv_SpM_spec;
```

```
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

◀BACK

05.5.1

102

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 temp frame buffers as ODSSs
Recv_SpM_spec.build_regist_info_ODSS (Temp_Audio_Buffer.GetId(), RW);
Recv_SpM_spec.build_regist_info_ODSS (TempAudioBufferInPeriod.GetId(), RW);
//register Application-Workspace-buffer as ODSS
Recv_SpM_spec.build_regist_info_ODSS (m_SPK.GetId(), RW);

// register SpM
if (RegisterSpM ( (PFSpMBody) Recv_SpM, &Recv_SpM_spec) == FAIL)
    TMOSLprintf ("Fail to register Recv_SpM object \n");
}
```

Recv_SpM
registration

05.5.1 103

UCI
DREAM Lab



Receiver.cpp (con't)

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

```
int i;
int AudioMsgSize;
int result;
MicroSec timestamp;
MicroSec SpMStartTime = GetCurrentDCSage ();
tms OfficialReleaseTime;
static counter = 0;

unsigned int nTempAudioBufferIndex;
unsigned int nTempAudioBufferHead;
MicroSec aTempTPT;
```

// Temporary
variables used in
SpM

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

05.5.1 104

UCI
DREAM Lab



Receiver.cpp (con't)

```
// Check Temp_Audio_Buffer
```

```
if(!Temp_Audio_Buffer.IsAudioBufferEmpty ())
{
    nTempAudioBufferIndex =
    Temp_Audio_Buffer.GetAudioBufferTail ();
    nTempAudioBufferHead =
    Temp_Audio_Buffer.GetAudioBufferHead ();
    do
    {
        // Get the TPT of a frame
        aTempTPT =
            Temp_Audio_Buffer.GetTPTofAudioFrame
            (nTempAudioBufferIndex);
```



05.5.1

105




Receiver.cpp (con't)

```
// If TPT of a frame is very close to the current time, play it right now
```

```
if (aTempTPT <= GetCurrentDCSage() + AUDIO_DIRECT_PLAY_RANGE * 1000)
{
    Temp_Audio_Buffer.CopyFrame (&AudioFrame);
    Temp_Audio_Buffer.GetFrameIDofAudioFrame (nTempAudioBufferIndex);
    m_SPK.Play ( (char *) AudioFrame.aData, WAVEOUT_AUDIO_BLOCK_SIZE);
}
```

```
// If TPT of a frame is within the execution time of this SpM iteration, move it to
// another buffer: Temp_Audio_Buffer_In_Period.
```

```
else if (aTempTPT <= SpMStartTime + AUDIO_CIRCLE_LENGTH * 1000 +
(RECEIVER_SPM_PERIOD - AUDIO_CIRCLE_LENGTH) * 1000 / 2
    && aTempTPT > GetCurrentDCSage () + AUDIO_DIRECT_PLAY_RANGE * 1000)
{
    Temp_Audio_Buffer.CopyFrame (&AudioFrame);
    if (!TempAudioBufferInPeriod.WriteAudioFrame (&AudioFrame))
        TMOSLprintf ("empAudioBufferInPeriodFULL\n");
}
```

05.5.1

106




Receiver.cpp (con't)

If TPT of a frame is not within the execution time of this SpM iteration, keep this frame in the Temp_Audio_Buffer and wait until the next SpM iteration.

```

else if (aTempTPT > SpMStartTime + AUDIO_CIRCLE_LENGTH * 1000 +
(RECEIVER_SPM_PERIOD - AUDIO_CIRCLE_LENGTH) * 1000 / 2) {
    break;
}
nTempAudioBufferIndex = Temp_Audio_Buffer.GetAudioBufferTail ();
} while (nTempAudioBufferIndex != nTempAudioBufferHead);
}

```

05.5.1

107

UCI
DREAM Lab

Receiver.cpp (con't)

```

while (1)
{
    // Get frames from RMMC channel by calling NonBlocingReceive until returning
    // there is no more frame in the RMMC channel

```

```

result = RMMC_1.NonBlockingReceive ( (void *) &AudioFrame,
                                     &AudioMsgSize, OfficialReleasTime, timestamp);
if (result != SUCCESS)
{
    if (result == NO_VALUE)
    {
        TMOSLprintf (_T("NO_VALUE\n"));
    }
    else if (result == FAIL)
        TMOSLprintf (_T("FAIL\n"));
    break;
}
}

```

05.5.1

108

UCI
DREAM Lab

Receiver.cpp(con't)

```
else //Successfully get a frame from RMMC channel
{
    TMOSLprintf (_T("TPT %d\n"), AudioFrame.aFrameID);
    AudioFrame.aRecv = GetCurrentDCSage ();
```

```
    if (AudioFrame.aTPT <= GetCurrentDCSage () +
        AUDIO_DIRECT_PLAY_RANGE * 1000)
    {
        m_SPK.Play ( (char*) AudioFrame.aData,
                    WAVEOUT_AUDIO_BLOCK_SIZE);
    }
```

If TPT of a frame is very close to the current time, play it right now.

05.5.1

109

UCI
DREAM Lab

Receiver.cpp (con't)

If TPT of a frame is within the execution time of this SpM iteration, move it to the buffer: Temp_Audio_Buffer_In_Period.

```
else if (AudioFrame.aTPT <= SpMStartTime + AUDIO_CIRCLE_LENGTH * 1000 +
        (RECEIVER_SPM_PERIOD- AUDIO_CIRCLE_LENGTH) * 1000 / 2
        && AudioFrame.aTPT > GetCurrentDCSage() +
        AUDIO_DIRECT_PLAY_RANGE * 1000)
{
    if (!TempAudioBufferInPeriod.WriteAudioFrame(&AudioFrame))
        TMOSLprintf("TempAudioBufferInPeriodFULL\n");
}
```

```
else if (AudioFrame.aTPT > SpMStartTime + AUDIO_CIRCLE_LENGTH * 1000 +
        (RECEIVER_SPM_PERIOD - AUDIO_CIRCLE_LENGTH) * 1000 / 2)
{
    if (!Temp_Audio_Buffer.WriteAudioFrame (& AudioFrame))
        TMOSLprintf ("Temp_Audio_BufferFULL\n");
}
```

If TPT of a frame is not within the execution time of this SpM iteration, keep this frame in the Temp_Audio_Buffer and wait until the next SpM iteration.

05.5.1

110

UCI
DREAM Lab

Receiver.cpp (con't)

```
bool bAudioBufferEmpty = false;
while (GetCurrentDCSage () <= SpMStartTime + CircleLength * 1000)
{
```

◀BACK

//Check Temp_Audio_Buffer_In_Period within this SpM iteration

```
if (!bAudioBufferEmpty && (GetCurrentDCSage() <= SpMStartTime +
    AUDIO_CIRCLE_LENGTH * 1000))
{
    if (!TempAudioBufferInPeriod.IsAudioBufferEmpty ())
    {
        nTempAudioBufferIndex =
            TempAudioBufferInPeriod.GetAudioBufferTail ();
        nTempAudioBufferHead =
            TempAudioBufferInPeriod.GetAudioBufferHead ();
        do
        {
            aTempTPT =
                TempAudioBufferInPeriod.GetTPTofAudioFrame
                    (nTempAudioBufferIndex);
```

05.5.1

111

UCI
DREAM Lab



Receiver.cpp (con't)

// If TPT of a frame is very close to the current time, play it right
// now.

```
if (aTempTPT <= GetCurrentDCSage () +
    AUDIO_DIRECT_PLAY_RANGE * 1000) {
    TempAudioBufferInPeriod.CopyFrame (& AudioFrame);
    m_SPK.Play ( (char *) AudioFrame.aData,
        WAVEOUT_AUDIO_BLOCK_SIZE);
}
```

```
nTempAudioBufferIndex =
    TempAudioBufferInPeriod.GetAudioBufferTail ();
} while (nTempAudioBufferIndex != nTempAudioBufferHead);
}
else bAudioBufferEmpty = true;
}
if (bAudioBufferEmpty) break;
}
SpM_iteration++;
return 1;
}
```

05.5.1

112

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 ();

    TMO_RegistParam TMO_Recv_spec;
    _tcscopy (TMO_Recv_spec.global_name, TMO_name);
    TMO_Recv_spec.start_time = start_time;

    //Register TMO
    RegisterTMO (& TMO_Recv_spec);
};
```

[◀BACK](#)

05.5.1

113

**UCI
DREAM Lab**