Real-time audio signal capture and processing using MATLAB object oriented programming

Samarth Hosakere Shivaswamy, Xiang Zhou, Stephen Roessner, Cheng Shu, Dave Headlam, and Mark Bocko

Citation: Proc. Mtgs. Acoust. **18**, 025006 (2012); doi: 10.1121/1.4794857
View online: https://doi.org/10.1121/1.4794857
View Table of Contents: http://asa.scitation.org/toc/pma/18/1
Published by the Acoustical Society of America

Proceedings of Meetings on Acoustics

Volume 18, 2012 http://acousticalsociety.org/

164th Meeting of the Acoustical Society of America

Kansas City, Missouri

22 - 26 October 2012

Session 2aED: Education in Acoustics

2aED11. Real-time audio signal capture and processing using MATLAB object oriented programming

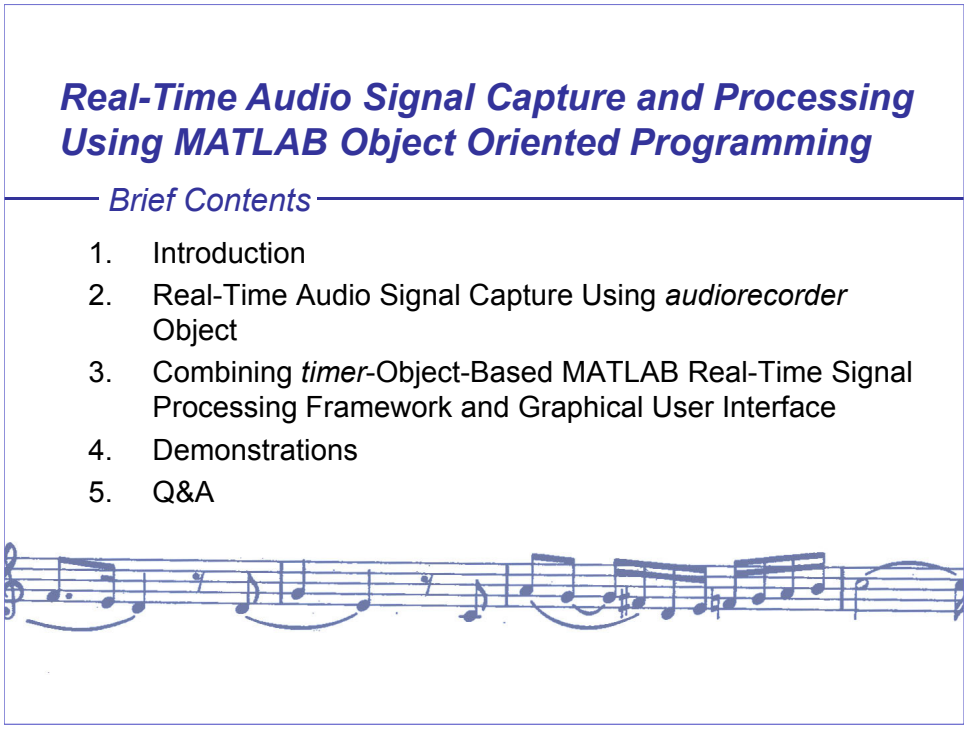
Samarth Hosakere Shivaswamy*, Xiang Zhou, Stephen Roessner, Cheng Shu, Dave Headlam and Mark Bocko

***Corresponding author's address: Dept. of Electrical and Computer Engineering, Edmund A. Hajim School of Engineering and Applied Sciences, University of Rochester, Dept. of Electrical and Computer Engineering, Rochester, NY 14627, sshivasw@z.rochester.edu**

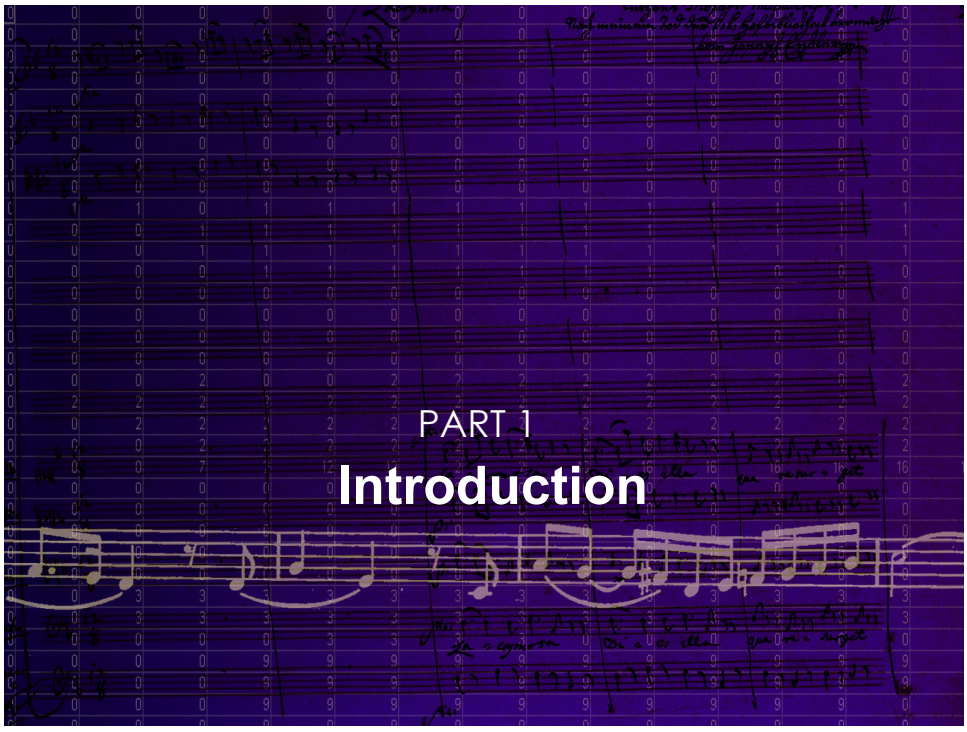
In MATLAB programming language the real-time audio processing functions are usually simulated in non-real-time due to a lack of real-time audio programming support. As a result the real-time audio signal capture and processing functionalities are usually implemented in other programming languages and cannot utilize the extensive signal processing functionalities provided by MATLAB. In this paper we introduce a MATLAB real-time signal processing framework based on MATLAB timer object and audiorecorder object. The proposed processing framework serves as an alternative solution for real-time programming implementation and demonstration. In our proposed processing framework the timer object is implemented to handle the looping of processing cycle, schedule the signal processing tasks, and handle the error processing. The audio capturing/processing functionality is implemented in the timer cycle by using two audiorecorder objects that read the audio streaming data and feed a segment of data to signal processing alternatively. The proposed framework achieves satisfactory real-time performance with no missing audio frames when a short audio delay setting of 10ms is applied. Several application examples of our proposed framework are also demonstrated.

Published by the Acoustical Society of America through the American Institute of Physics

 [DOI: 10.1121/1.4794857]
Received 3 Feb 2013; published 25 Feb 2013



We first provide a introduction on the background of this project. Then we illustrate this framework in part 2, and 3, with details in system and MATLAB implementation. We also perform a demonstration.



In Part 1 we introduce the basic concept of this project.

PART 1 Introduction

MATLAB, Signal Processing, and Audio

- MATLAB provides extensive digital signal processing functionalities, most of them is applicable for audio signal processing.
- Extensive MATLAB-based materials/codes, both at instruction level and research level, are available for both general areas of signal processing and audio signal processing.
- Object oriented programming (OOP) [1,2], as an advanced topic in MATLAB programming, is usually hard for beginners. This proposed framework introduces OOP concepts and practice in a nice interactive context. The OOP concept in this framework is important for advanced MATLAB graphics. The same concepts are applicable to other programming framework and enable students to tackle and construct large-scale “real-world” projects.
- Audio provide a nice auralization interface that compliment conventional visualization techniques. Auralization is important for demonstrating signal processing concepts: students can hear the results at each processing stage. The overhead for learning to implement basic audio signal processing functionalities is much less compared with image/video based studies.

MATLAB, signal processing, and audio form an integrated teaching framework. The hard concepts of signal processing and advanced audio programming can be better instructed in the application environment of audio engineering.

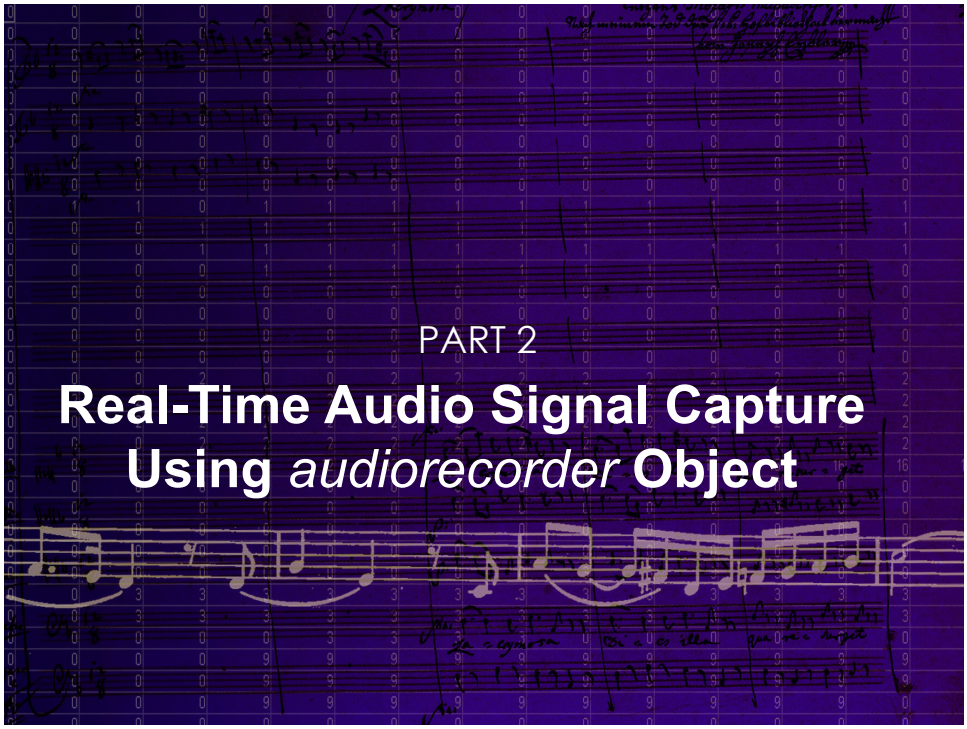
PART 1 Introduction

The Need for Real-Time

- A general purpose MATLAB based framework for real-time audio signal processing is not available. To perform real-time signal processing functionalities, special hardware/programming-language have to be utilized.
- Real-time processing programs provide motivating experience for our students. The resulting programs is similar to signal processing/analysis functionalities of real-world audio software.
- The proposed real-time processing framework can be easily integrated with our existing instruction recording studio, production facilities and classroom demonstrations.
- Compare to MATLAB signal acquisition toolbox, simulink and other hardware-based methods, students can simply change the code segment and run this proposed framework. This framework is also easier to implement in the beginner audio engineering courses.

References:
[1] S. T. Smith, MATLAB Advanced GUI Development, Dog Ear Publishing, 2006.
[2] A. H. Register, A Guide to MATLAB Object-Oriented Programming, Chapman and Hall, 2007.

Real-time audio processing is challenging if directly programmed from MATLAB. However, the implementation of this framework in educational setting is rewarding.



In Part 2, we illustrate how to dynamically capture audio signal using an "object" in MATLAB called audiorecorder.

PART 2 Real-Time Audio Signal Processing

Basic *audiorecorder* Functionalities

- Create audio recorder object:

```
a_obj_handle = audiorecorder(sample_rate,number_of_bits,
channels_number);
```

A more complicated example:

```
hexchangedata.recorder_1 =
audiorecorder(hexchangedata.SAMPLE_RATE_1,8,1);
```

- Record audio:

```
record(a_obj_handle);
record(hexchangedata.recorder_1);
```

- Stop recording and obtain audio data:

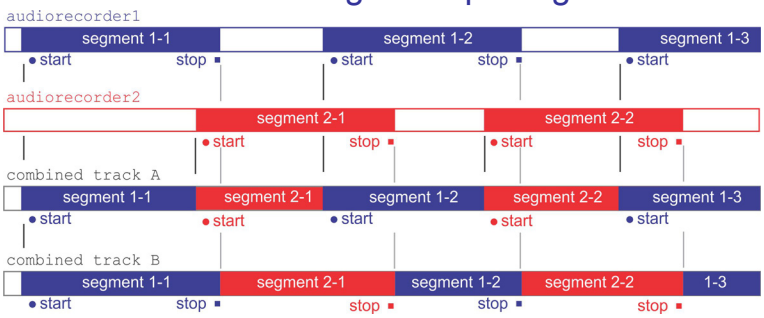
```
stop(a_obj_handle);
mydata = getaudiodata(a_obj_handle);

stop(hexchangedata.recorder_1);
hexchangedata.analysisdata =
getaudiodata(hexchangedata.recorder_1);
```

We can perform various operations on a audiorecorder object. After creating the object, we can record, stop, play and retrieve the audio data.

PART 2 Real-Time Audio Signal Processing

Using Multiple *audiorecorder* Objects for Real-Time Audio Signal Capturing



- By alternatively starting and stoping two *audiorecorder* objects, we can extract real-time audio frames at the stop points and perform analysis.
- If each "stop" occurs after the "start" of another *audiorecorder*, the complete audio signal is captured and can be stitched together using timing information recorded at each "start" position.
- For smaller frames, we can employ more *audiorecorder* objects so that the start/stop operations will not affect the completeness of audio signal capturing.

In this example, combined track B is better because the audio samples near the "stop" positions are kept. This option makes the audio frames more "current".

PART 2 Real-Time Audio Signal Processing

More Related Functionalities

- Create audio recorder object in the "global" domain so it can be accessed at multiple places:

```
global hexchangedata;
```

Define the recorder in a manner of "object oriented programming", and inherit the domain of "hexchangedata":

```
hexchangedata.recorder_1 =
audiorecorder(hexchangedata.SAMPLE_RATE_1,8,1);
```

- Obtain the timing of each audio frame:

```
tic; % start the watch
mytime = toc;

tic
hexchangedata.currenttime = toc;
hexchangedata.previousetimea =
                hexchangedata.currenttime;
```

The timing data of each captured frame is obtained using "tic" and "toc".

PART 2 Real-Time Audio Signal Processing

Coordinate Two *audiorecorder* Objects

- Coordinating two *audiorecorder* objects, and tracking the time point:

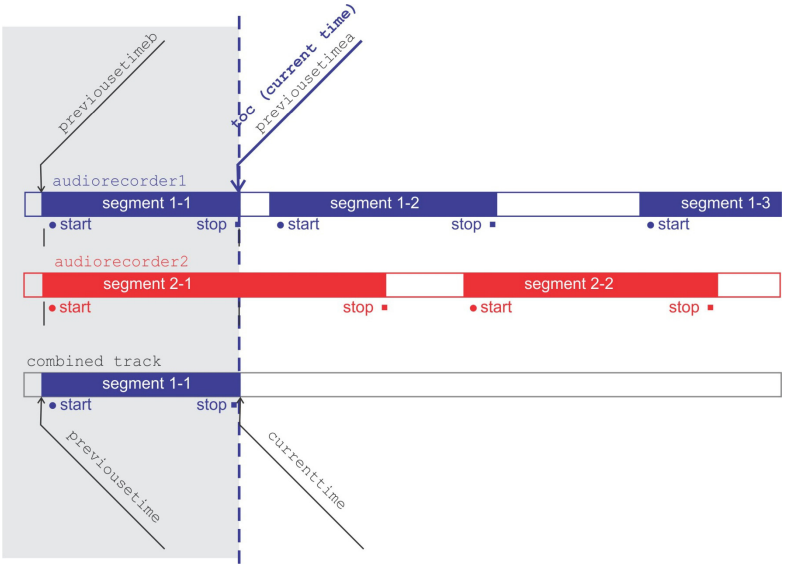
```
% stop two recorders alternatively
if rem(hexchangedata.count , 2) == 1
  if isrecording(hexchangedata.recorder_1) == 1
     hexchangedata.previousetime =
             hexchangedata.previousetimeb;
     stop(hexchangedata.recorder_1);
     hexchangedata.currenttime = toc;
     hexchangedata.previousetimea =
               hexchangedata.currenttime;
     hexchangedata.analysisdata =
              getaudiodata(hexchangedata.recorder_1);
  end
  % begin recording again
  record(hexchangedata.recorder_1);
end
```

We employed two audiorecorder objects so we can capture all audio signals. Then we use a data structure to track the time for each frame we captured.

PART 2 Real-Time Audio Signal Processing

Coordinate Two *audiorecorder* Objects

- Coordinating two *audiorecorder* objects, and tracking the time point:



This is the illustration of time-tracking mechanism in the first step. We record the start time and stop time for the frame we captured. This is the step 1 of audio capture processing.

PART 2 Real-Time Audio Signal Processing

Coordinate Two *audiorecorder* Objects

- Coordinating two *audiorecorder* objects, and tracking the time point:
  [Code for recorder 2]

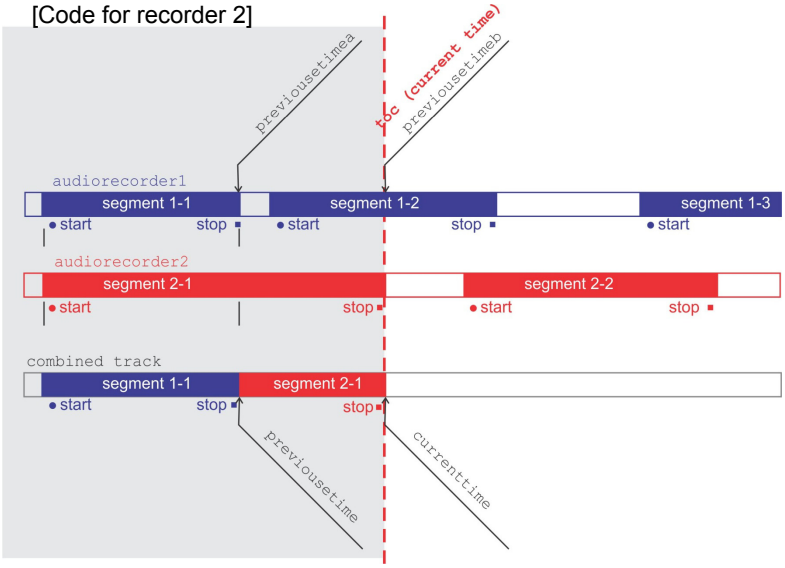
```
if rem(hexchangedata.count , 2) == 0
    if isrecording(hexchangedata.recorder_2) == 1
       hexchangedata.previousetime =
                  hexchangedata.previousetimea;
       stop(hexchangedata.recorder_2);
       hexchangedata.currenttime = toc;
       hexchangedata.previousetimeb =
                 hexchangedata.currenttime;
       hexchangedata.analysisdata =
                 getaudiodata(hexchangedata.recorder_2);
     end
     % begin recording again
     record(hexchangedata.recorder_2);
end
```

Code segment for the step-2 part of the program. We start another recorder and get more audio streams. Then we iterate over step 1 and 2.

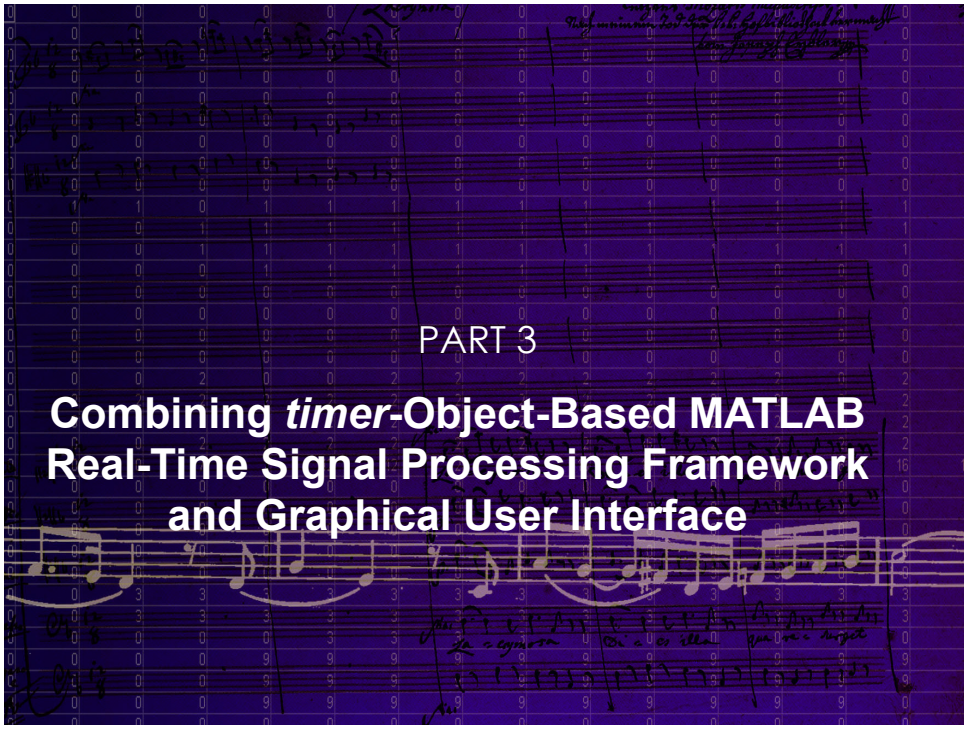
PART 2 Real-Time Audio Signal Processing

Coordinate Two *audiorecorder* Objects

- Coordinating two *audiorecorder* objects, and tracking the time point:
  [Code for recorder 2]



The result of performing step 2. We changed the data in time-tracking data structure. The new frame we add in will not overlap with previous frame. In the same time we have a complete audio signal (no missing part).



We can iterate the tools in Part 2 to program a prototype processing framework. In Part 3 we further introduce MATLAB real-time processing and GUI to provide better control and user interaction.

PART 3 Combining MATLAB Real-Time Framework

Timer Object based MATLAB Real-Time Processing Framework

- A timer object [1,2] is implemented to handle the looping operation and schedule the subsequent processing operations. Here we create a *timer* object, with a update rate of 0.1 second:

```
hexchangedata.timer = timer('TimerFcn',...
 {@timer_Callback, hObject},...
'ErrorFcn',{@timererr_Callback, hObject},...
'ExecutionMode', 'fixedRate','Period', 0.1, ...
'BusyMode','error');
```

  In practical implementation we have two timer objects because we want to do two independent tracks of visualization.

- Regular timer session:

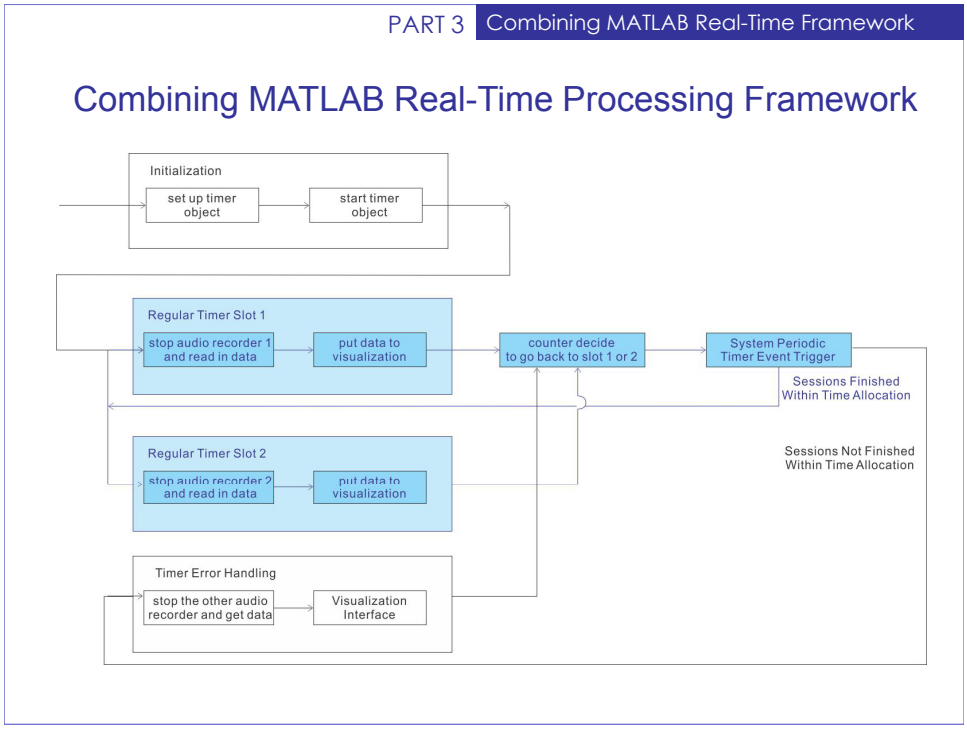
```
function timer_Callback(timer_object, eventdata,
hObject)
```

- Error-handling session:

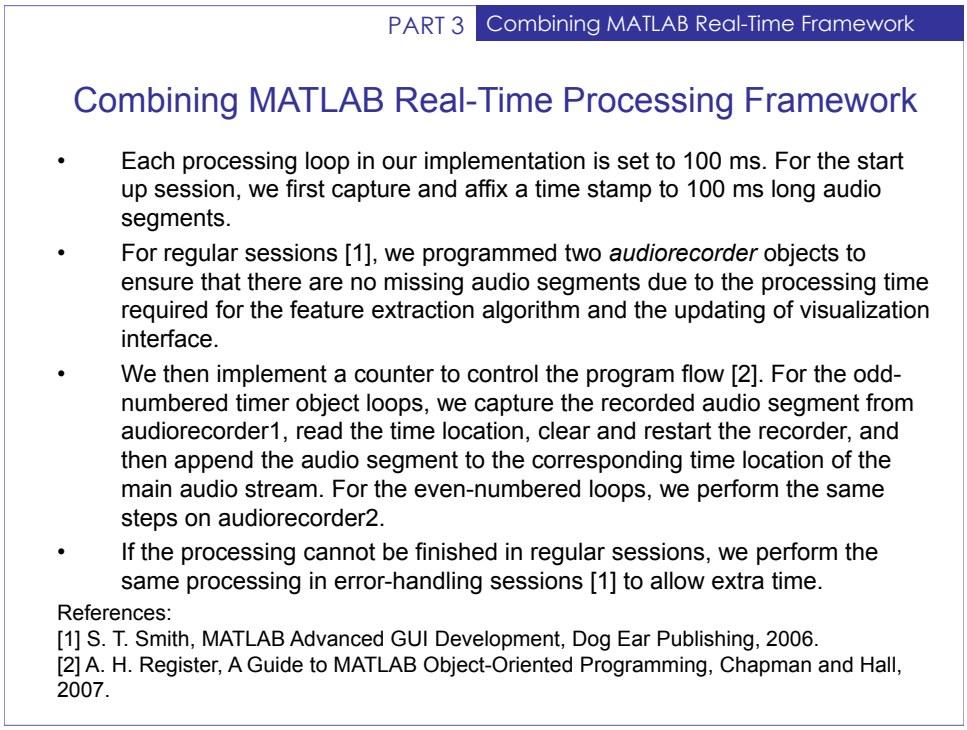
```
function timererr_Callback(timer_object, eventdata,
hObject)
```

[1] S. T. Smith, MATLAB Advanced GUI Development, Dog Ear Publishing, 2006.
[2] A. H. Register, A Guide to MATLAB Object-Oriented Programming, Chapman and Hall, 2007.

The MATLAB real-time processing framework is based on “timer” object. A “timer” object coordinates the processing comments in an iteration block.



If the block is completed in the “timer” slot (0.1second here), the next regular session will run after this slot. If slot time is exceeded, error-handling is activated.



If the processing cannot be finished in a “timer” slot, we activate the error-handling session. In an error-handling session we carry the same comments, but allows extra time for these operations to finish.

PART 3 Combining MATLAB Real-Time Framework

Combining MATLAB Real-Time Processing Framework

- Programming regular sessions:

```
function timer_Callback(timer_object, eventdata, hObject)
```

implement a counter to control recorder 1 and 2:

```
hexchangedata.count = hexchangedata.count + 1;
```

stop recorders alternatively to get the audio data

```
if rem(hexchangedata.count , 2) == 1
  if isrecording(hexchangedata.recorder_1) == 1
  hexchangedata.previousetime =
          hexchangedata.previousetimeb;
  stop(hexchangedata.recorder_1);
  hexchangedata.currenttime = toc;
  hexchangedata.previousetimea =
           hexchangedata.currenttime;
  hexchangedata.analysisdata =
           getaudiodata(hexchangedata.recorder_1);
 end
```

start recorder again, and perform visualization tasks

```
 record(hexchangedata.recorder_1);
end
```

To program a regular session, we need to create a “timer” object and then put audio functions in its sessions. The “timer” object comes with several callback-functions that controls the program flows.

PART 3 Combining MATLAB Real-Time Framework

Combining MATLAB Real-Time Processing Framework

- Programming error-handling sessions:

```
function timererr_Callback(timer_object, eventdata,
hObject)
```

implement a counter to control recorder 1 and 2:

```
hexchangedata.count = hexchangedata.count + 1;
```

stop recorders alternatively to get the audio data

```
if rem(hexchangedata.count , 2) == 1
  if isrecording(hexchangedata.recorder_1) == 1
   … … … [Same with regular sessions]
           hexchangedata.currenttime;
  hexchangedata.analysisdata =
           getaudiodata(hexchangedata.recorder_1);
 end
```

start recorder again, and perform visualization tasks

```
 record(hexchangedata.recorder_1);
end
```

start regular timer session:

```
start(hexchangedata.timer);
```

The time allocation in error-handling session is not limited to regular time slot. This buffering mechanism prevents losing audio frames.

PART 3 Combining MATLAB Real-Time Framework

Combines *timer*-based Real-Time Processing Framework and Graphical User Interface

- Provide basic programming data:

```
function expreal_OpeningFcn(hObject, eventdata, handles,
varargin)
```

The following instructions will be executed before is GUI is visible. Here we initialize the *timer* object, *audiorecorder* objects, and programming the GUI.

- Programming a "record" button.

```
function recordaudiosrc_Callback(hObject, eventdata,
handles)
```

start the "stop watch" and the recorders:

```
tic;
record(hexchangedata.recorder_1);
```

start the timer:

```
start(hexchangedata.timer);
```

The two-recorder audio capture and real-time processing can be encoded into a graphical user interface. First we put initialization code in "OpeningFcn" (open function). Then we program each buttons in the user interface.

PART 3 Combining MATLAB Real-Time Framework

Combines *timer*-based Real-Time Processing Framework and Graphical User Interface

- Programming a "stop" button.

```
function stopbutton_Callback(hObject, eventdata, handles)
```

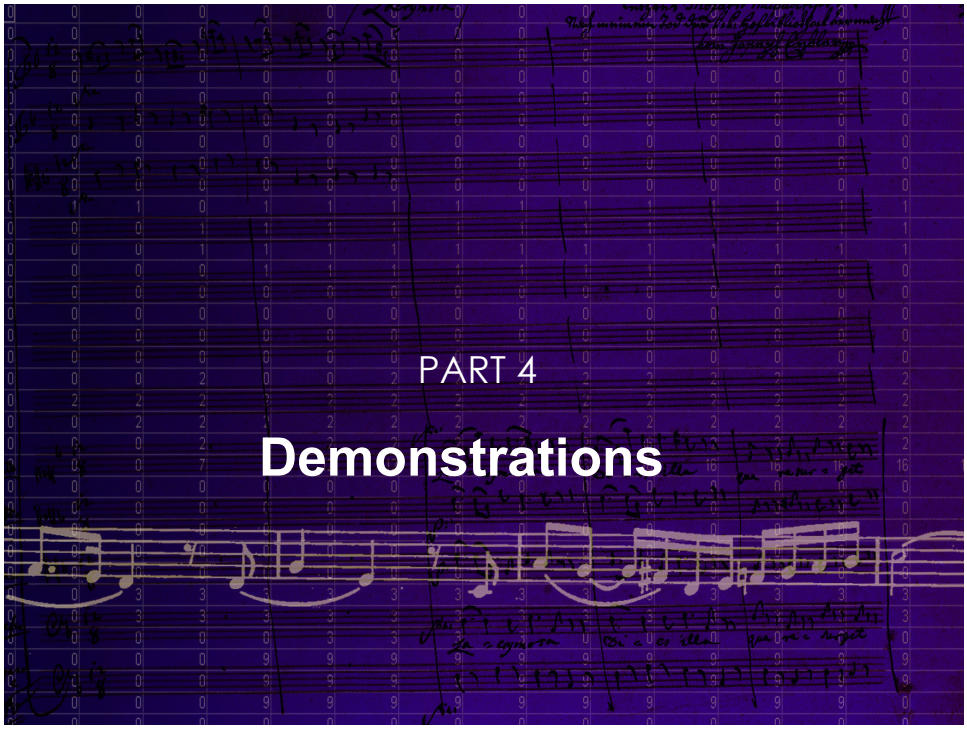
stop the active *timer* object:

```
stringdetec = get(hexchangedata.timer,'Running');
if strcmp(stringdetec, 'on')== 1
stop(hexchangedata.timer);
end
```

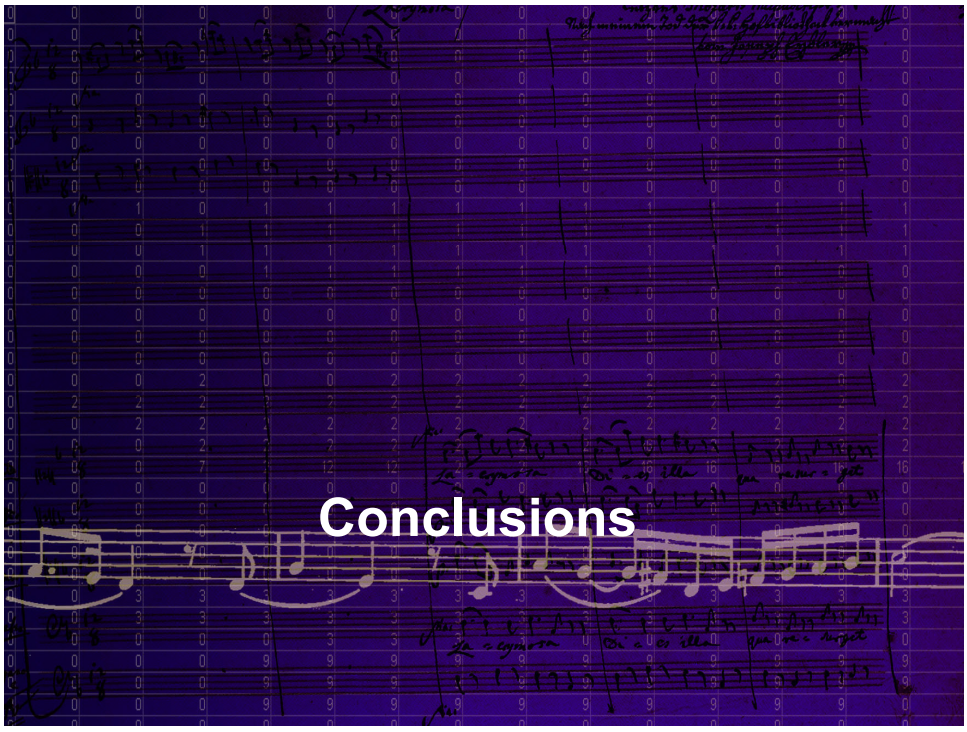
stop the active recorder objects:

```
if isrecording(hexchangedata.recorder_1) == 1
stop(hexchangedata.recorder_1);
end
if isrecording(hexchangedata.recorder_2) == 1
stop(hexchangedata.recorder_2);
end
```

The main function here is similar to a error-handling session of a "timer" object: we stop active "timer" and "audiorecorder", then get the data and finish the remaining processing frame.



This demonstration is based on real-time visualization of musical sound. The real-time processing is applied both to audio capturing and MATLAB graph.



This framework effectively converges audio, signal processing and software programming. As an educational tools this framework provides important hand-on preparations before more advanced topics.