# User’s Guide:
# How to Run C/MATLAB Codes on iOS Smartphones as Open Source and Portable Research Platforms for Hearing Improvement Studies

N. KEHTARNAVAZ AND A. SEHGAL

UNIVERSITY OF TEXAS AT DALLAS

FEB 2017

This work was supported by the National Institute of the Deafness and Other Communication Disorders (NIDCD) of the National Institutes of Health (NIH) under the award number 1R01DC015430-01. The content is solely the responsibility of the authors and does not necessarily represent the official views of the NIH.

## Table of Contents

# Introduction

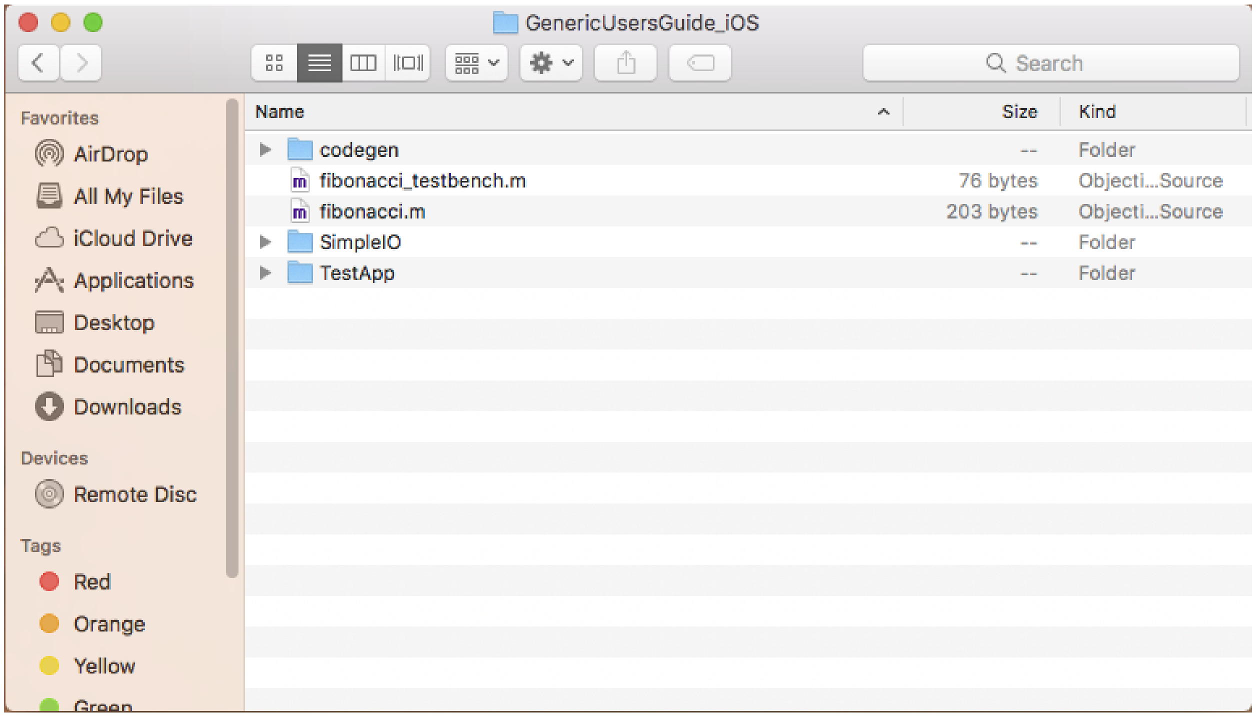
This user's guide covers the steps one needs to take in order to run C/MATLAB algorithms on iOS mobile devices (iPhones and iPads) as open source and portable research platforms for hearing improvement studies.

The first section covers the software tools required for algorithm implementation on iOS devices. In the second section, it is shown how to run C codes on iOS devices. The third section discusses the use of the MATLAB Coder for converting MATLAB codes into C codes. The fourth section covers hardware dependencies that users will need to be aware of when implementing algorithms on different iOS devices having different i/o characteristics. Finally, the fifth section describes frame-based processing for real-time operation.

## User's Guide Accompanying Codes

The accompanying codes for this user's guide consist of the following (see Fig. 1):

- "codegen" – This folder includes the code generated by the MATLAB coder for the MATLAB script "fibonacci.m".
- "fibonacci_testbench.m" – This file is the testbench MATLAB script associated with "fibonacci.m".
- "FrequencyDomain" – This folder includes the code solution for section 5.
- "TestApp" – This folder includes the code solution for sections 2 and 3.



| Name | Size | Kind |
|---|---|---|
| codegen | -- | Folder |
| fibonacci_testbench.m | 76 bytes | Objecti...Source |
| fibonacci.m | 203 bytes | Objecti...Source |
| SimpleIO | -- | Folder |
| TestApp | -- | Folder |

*Fig. 1*

## Section 1: Software Tools

iOS is the operating system developed by the company Apple for mobile devices such as iPhones, iPads and iPods. This operating system uses Objective-C. Here, an Objective-C shell is thus developed to allow running C codes within the iOS environment.

To implement C codes as apps on iOS devices, a Mac OS machine is needed. The Xcode IDE (Integrated Development Environment) software package needs to be installed on the machine. Xcode is designed by Apple for app development. It is available as a free download on the Mac App Store. It should be noted that Apple does not allow development of iOS apps from Windows or Linux based computers.

From iOS version 9 and Xcode version 7 onwards, it has become possible to perform app development without the need to enroll or register in the Apple Developer Program. It is worth mentioning that although apps developed can be run on iPhones/iPads, they cannot be published on the App Store without registering as an Apple Developer.

For help with the installation of Xcode IDE, the reader can refer to Chapter 3 of the book "Smartphone-Based Real-Time Digital Signal Processing", which can be acquired from the link:

http://www.morganclaypool.com/doi/abs/10.2200/S00666ED1V01Y201508SPR013

# Section 2: Running C Codes as iOS apps

## 2.1 Programming Language

For creating iOS apps, Objective-C is used to create the required shell. Objective-C constitutes a superset of C, allowing one to seamlessly call C functions by just importing a header file. The execution of C codes occurs efficiently within the Objective-C environment due to the absence of any overhead translation or matching.

## 2.2 Creating Objective-C Shell

The creation of an Objective-C shell starts by creating a GUI to link data to a C code. The steps needed for creating a basic shell are listed below:

- Open Xcode and select "Create a new Xcode project" on the startup splash screen, see Fig. 2. In case no splash screen comes up, select *File->New->Project* or press Command + Shift + n.

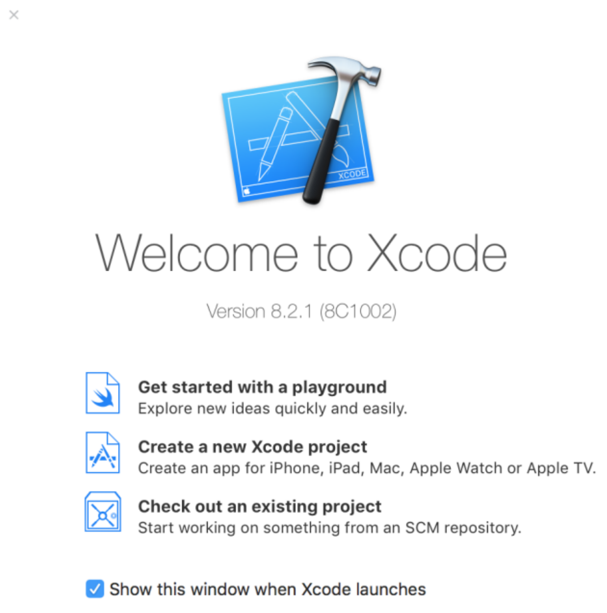

Welcome to Xcode
Version 8.2.1 (8C1002)
Get started with a playground
Explore new ideas quickly and easily.
Create a new Xcode project
Create an app for iPhone, iPad, Mac, Apple Watch or Apple TV.
Check out an existing project
Start working on something from an SCM repository.
Show this window when Xcode launches

*Fig. 2*

- On the page that comes up, shown in Fig. 3, choose the template of the project as "Single View Application" and click Next.

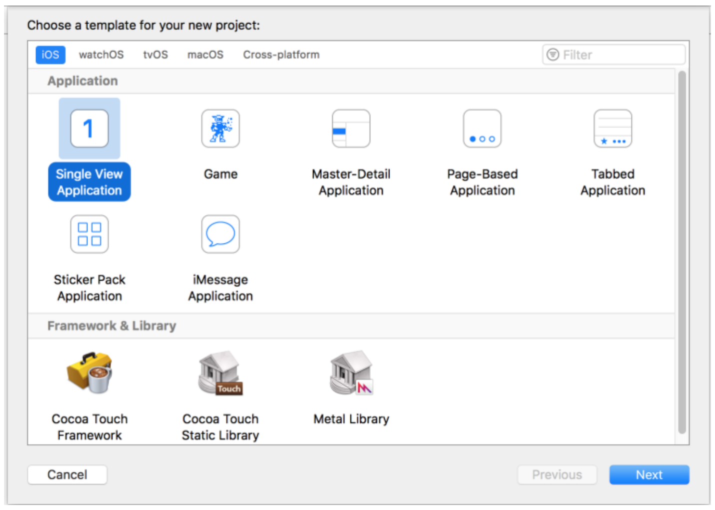
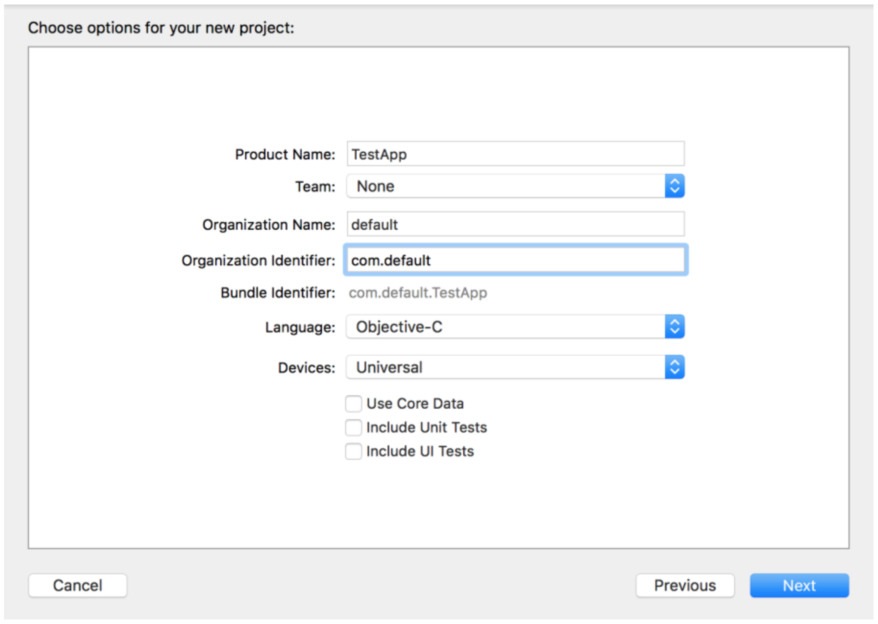


*Fig. 3*

- For options that come up, shown in Fig. 4, set the name of the project to “TestApp”, select Team as “None”, set your organization name as “default” and the organization identifier as “com.default”. Remember to set the Language to “Objective-C” as this option cannot be reversed. Select devices as “Universal” and deselect all the options. Then, click on Next. Note that these settings are dummy settings for a Test app. For an actual implementation, properly set the organization name and organization identifier.


Choose options for your new project:

| | |
|---|---|
| Product Name: | TestApp |
| Team: | None |
| Organization Name: | default |
| Organization Identifier: | com.default |
| Bundle Identifier: | com.default.TestApp |
| Language: | Objective-C |
| Devices: | Universal |

☐ Use Core Data
☐ Include Unit Tests
☐ Include UI Tests

*Fig. 4*

- Store the project in the Directory of your choice and click Create. If desired, deselect the option “Create Git Repository”.
- To be able to build the app for iPhones, you would need to sign the application and select a Team. In Xcode, select “User Name (Personal Team)” for Team and Xcode will automatically sign the application.

## 2.3 Creating GUI

- Navigate to the file “ViewController.m" in the file navigator, as shown in Fig. 5.

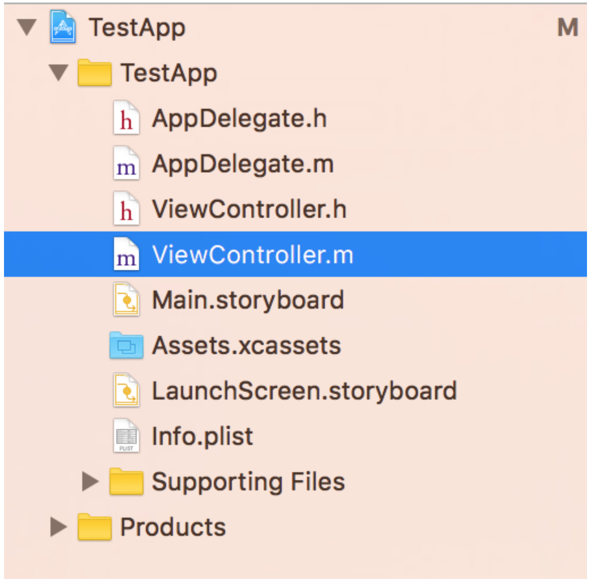


*Fig. 5*

- In the ViewController interface, declare the following two properties “UILabel” and “UIButton” as follows:

```
@interface ViewController ()
@property UILabel *label;
@property UIButton *button;
@end
```

- In the “viewDidLoad” method noted below in ViewController.m, one can initialize the declared properties and assign them to View. UIButton is assigned an action “buttonPress” programmatically via the following code:

```
- (void)viewDidLoad {
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.
    _label = [[UILabel alloc] initWithFrame:CGRectMake (10, 15, 300, 30)];
```

```
        _label.text = @"Hello World!";
        [self.view addSubview:_label];
        _button = [UIButton buttonWithType:UIButtonTypeRoundedRect];
        _button.frame = CGRectMake(10, 50, 300, 30);
        [_button setTitle:@"Button" forState:UIControlStateNormal];
        [self.view addSubview:_button];
        [_button addTarget:self action:@selector(buttonPress:)
          forControlEvents:UIControlEventTouchUpInside];
    }
    - (IBAction)buttonPress:(id)sender {

    }
```

The “buttonPress” method appears as a blank method here. A property will be assigned to it which will change the label and the text passed to it via the C code.

## 2.4 Adding C File

- To add a C file to the project, navigate to the “TestApp” folder in the project navigator and select “New File…”, shown in Fig. 6.

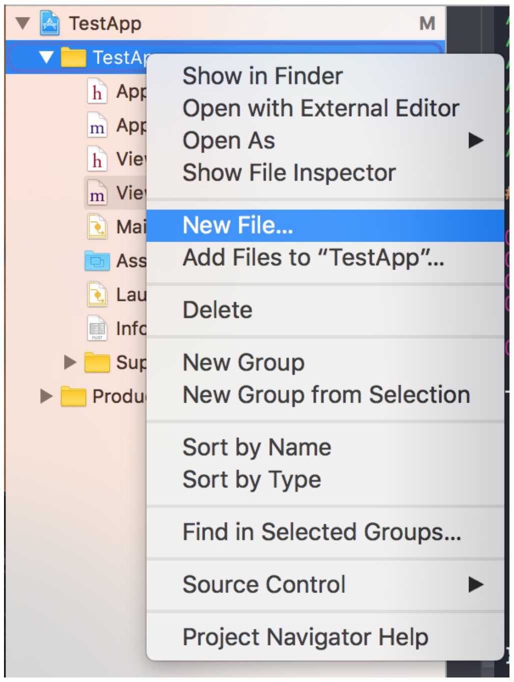


*Fig. 6*

- In the “iOS” tab, shown in Fig. 7, under “Source”, select “C File” and click Next.
- On the page that comes up, shown in Fig. 8, write the file name as “Algorithm” and also select the option that states “Also create a header file” and click Next.

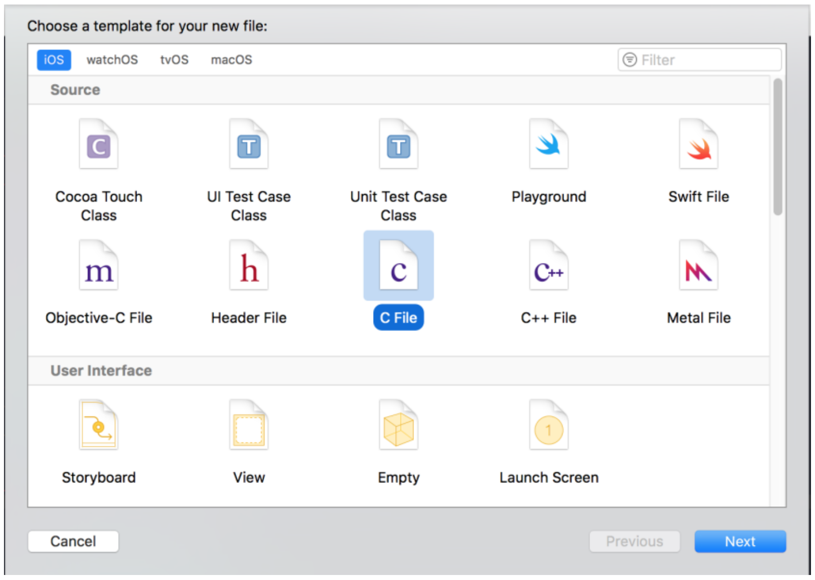
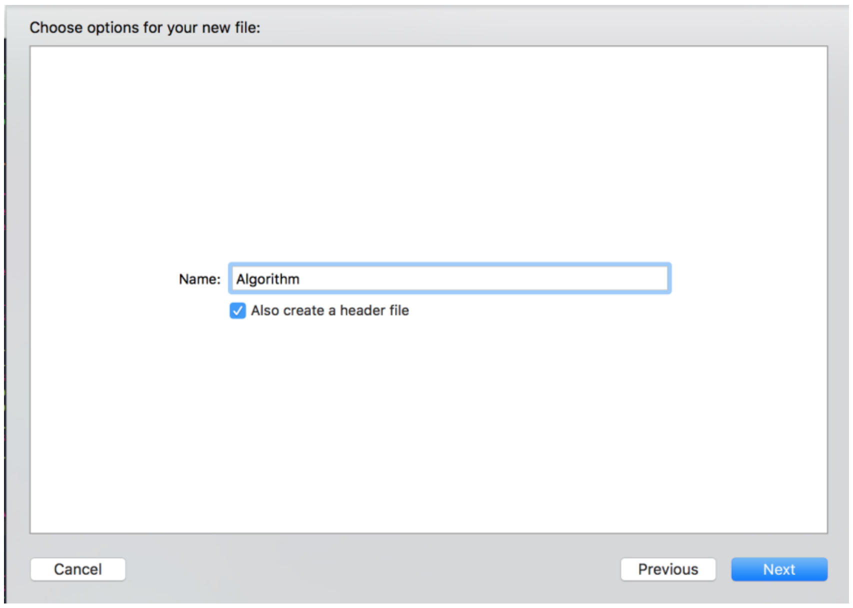


*Fig. 7*



Name: Algorithm

[✓] Also create a header file

*Fig. 8*

- Save the file in your project folder and hit Create. You will now be able to see two new files in your Project Navigator: Algorithm.c and Algorithm.h.
- In Algorithm.c, add the following simple C code:

```
const char *HelloWorld() {
        printf("Method Called");
        return "Hello World! called from C";
    }
```

This is a function that will pass back a string when called. Let us display the following string when the button is pressed in the GUI. To allow the function to be called in Objective-C, the function declaration needs to be placed in Algorithm.h. Add the following line of code before "#endif" in Algorithm.h:

```
const char *HelloWorld();
```

This function can now be called in Objective-C just by including the header file.

## 2.5 Calling C Functions in Objective-C

- In ViewController.m, just below #import "ViewController.h", add #import "Algorithm.h".
- In the "buttonPress" method, add the following line of code:

```
    _label.text = [NSString stringWithUTF8String:HelloWorld()];
```

- The code will alter the label displayed in the simulator upon execution.
- Run the program in the simulator and observe the output, illustrated in Fig. 9 and Fig. 10 before and after the button press, respectively:

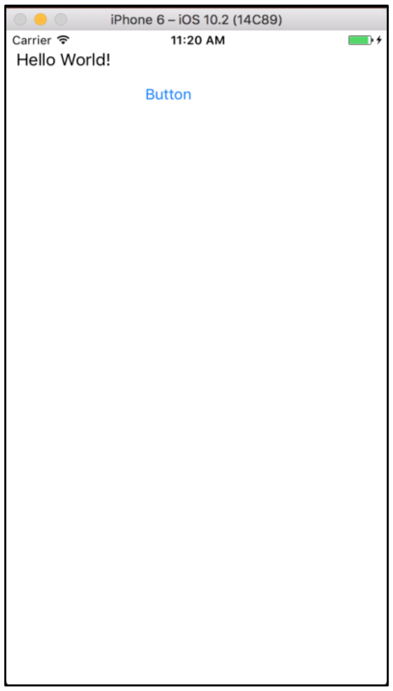


*Fig. 9*

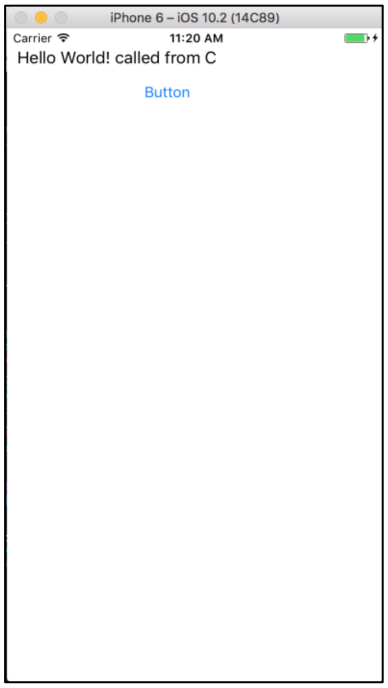
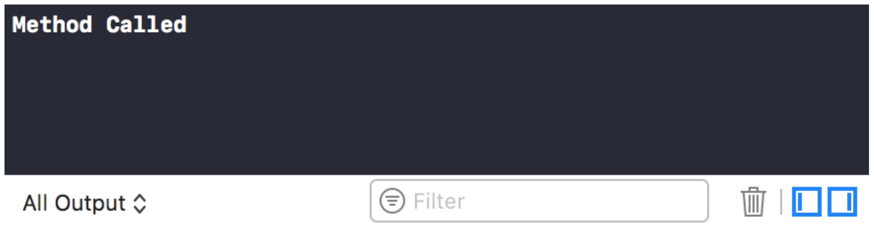


*Fig. 10*

Also in the Debug Console in Xcode, “Method Called” gets printed, see Fig. 11.



```
Method Called
```

*Fig. 11*

As noted above, C functions can be called in Objective-C in a seamless manner without the need to carry out any translation between the two languages.

# Section 3: Converting MATLAB Code to C

This section looks at deploying MATLAB functions on an iOS device using the MATLAB Coder utility of MATLAB. A simple example is shown here indicating the generation of C code from a MATLAB code which can be placed into the “TestApp” covered in section 2.

## 3.1 Creating a MATLAB Script

The MATLAB Coder requires a function of interest to be coded in MATLAB and also a Test Bench script to be coded in MATLAB for verifying the outputs of the function and also to see whether the function runs without any errors. The example shown here involves the creation of the Fibonacci sequence.

- Let us name a function “fibonacci” in MATLAB. Enter the following code. Keep the order of the parameters as noted below, otherwise the output will not appear correctly.

```
function fib_sequence = fibonacci(n,t1,t2)
fib_sequence = zeros(1,n);
fib_sequence(1) = t1;
fib_sequence(2) = t2;
for i = 3:n
   fib_sequence(i) = fib_sequence(i-1) + fib_sequence(i-2);.
end
```

- Next, create a test bench script to verify the output of the function. The test bench script for the above function is displayed below:

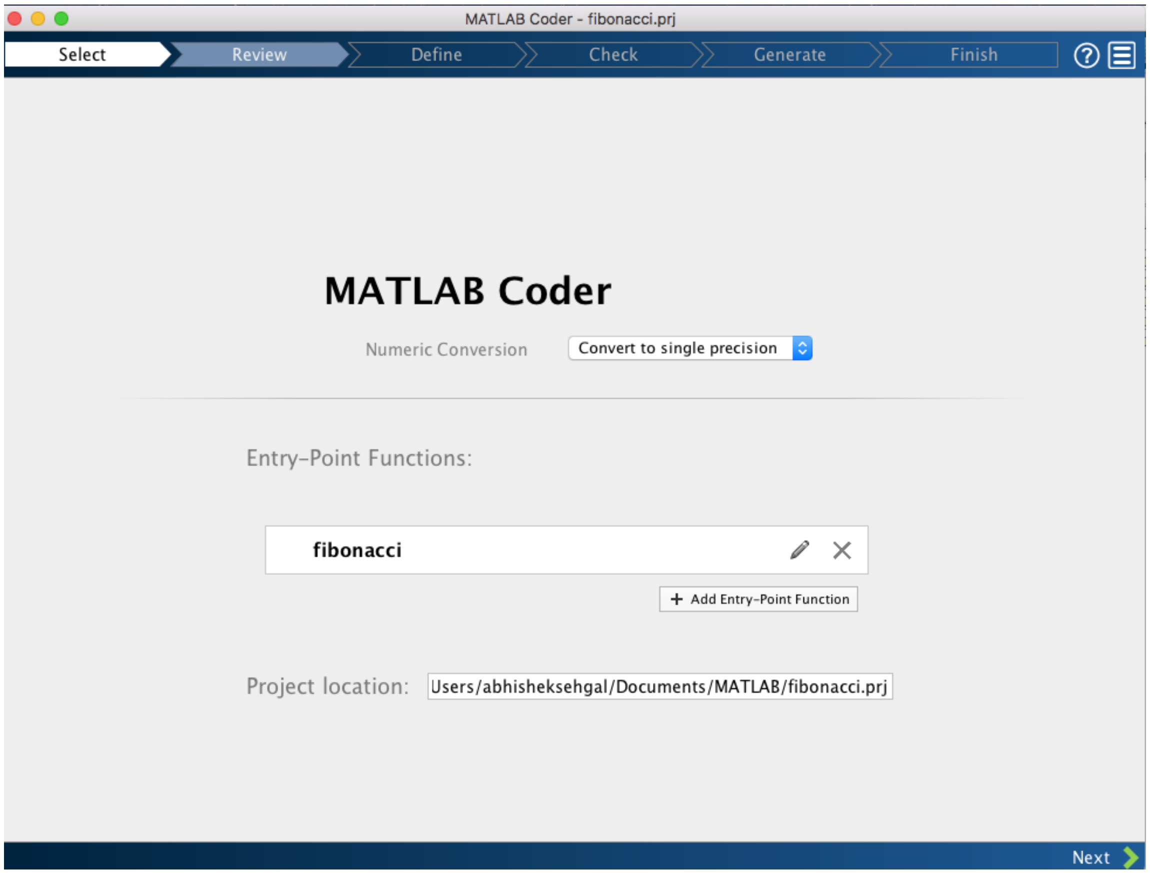
```
clear all;
clc;
n = 10;
t1 = 0;
t2 = 1;
fib_sequence = fibonacci(n, t1, t2);
```

This script allows generating the Fibonacci sequence for 10 numbers with the initial elements being 0 and 1.

## 3.2 Generating C Code Using MATLAB Coder

After the function and the test bench script have been created, an equivalent C code can be generated by using the MATLAB Coder. The MATLAB Coder can be found under the APPS tab in the MATLAB toolbar. Before starting the MATLAB Coder, make sure you are in the directory which contains the function and the test bench script.

- In the MATLAB Coder splash screen, shown in Fig. 12, select the Numeric Conversion as "Convert to single precision". In the Generate code for function, browse the function name. After you select both the options, the splash screen will display the location where the MATLAB Coder will create the project. Click Next.

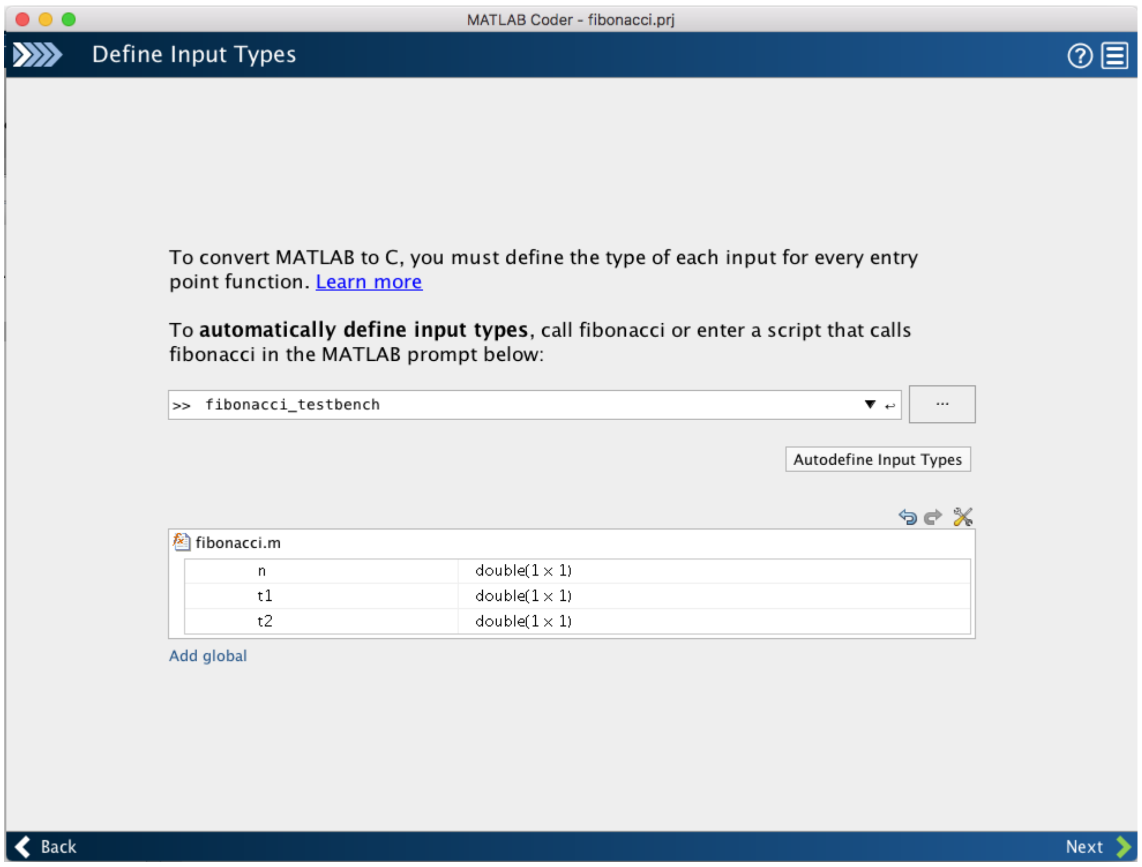


Numeric Conversion: Convert to single precision

Project location: Users/abhisheksehgal/Documents/MATLAB/fibonacci.prj

*Fig. 12*

- In the screen that comes up, shown in Fig. 13, select the test bench and click "Autodefine input Types". Make sure the option "Does this code use global variables?" is set to No. This will populate the splash screen with all the inputs to the function. Verify all the three inputs are present, namely n, t1 and t2. Then, click Next.



To convert MATLAB to C, you must define the type of each input for every entry point function. Learn more

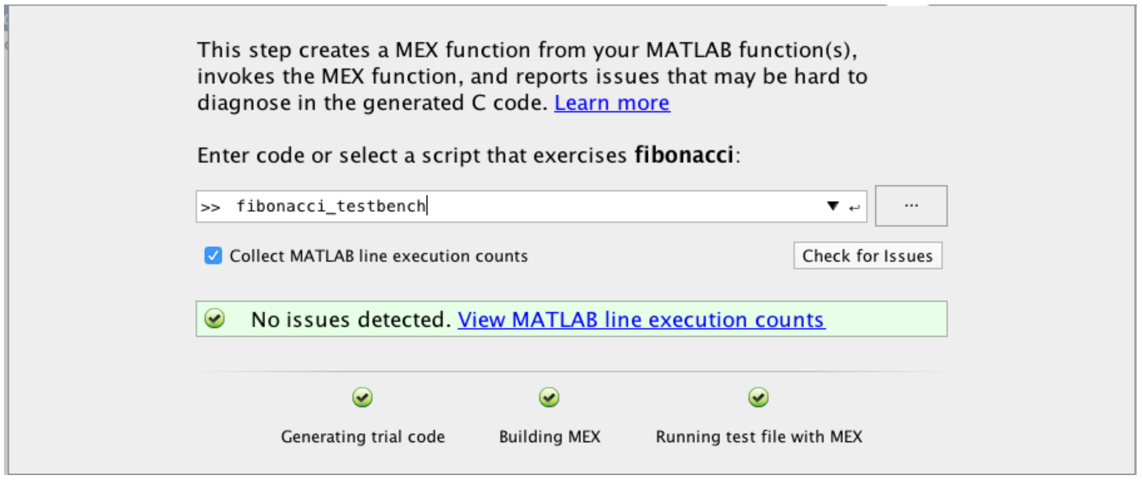
To **automatically define input types**, call fibonacci or enter a script that calls fibonacci in the MATLAB prompt below:

```
>> fibonacci_testbench
```

| | |
|---|---|
| n | double(1 × 1) |
| t1 | double(1 × 1) |
| t2 | double(1 × 1) |

*Fig. 13*

- In the next splash screen, shown in Fig. 14, click "Check for Issues". If any errors are displayed at this point, rectify them. Click Next when the screen shows "No issues detected".



This step creates a MEX function from your MATLAB function(s), invokes the MEX function, and reports issues that may be hard to diagnose in the generated C code. Learn more

Enter code or select a script that exercises **fibonacci**:

```
>> fibonacci_testbench
```

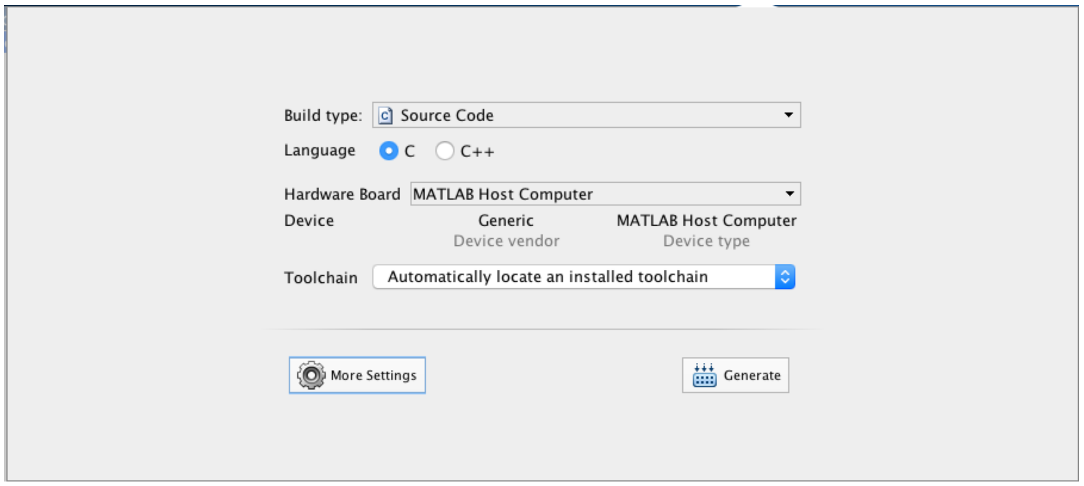
☑ Collect MATLAB line execution counts

No issues detected. View MATLAB line execution counts

*Fig. 14*

- In the next screen that comes up, shown in Fig. 15, check to see that the options appear as
  - Build Type: Source Code
  - Language: C

  If these options are correct, then click "Generate".



Build type: Source Code

Language [✓ C] C++

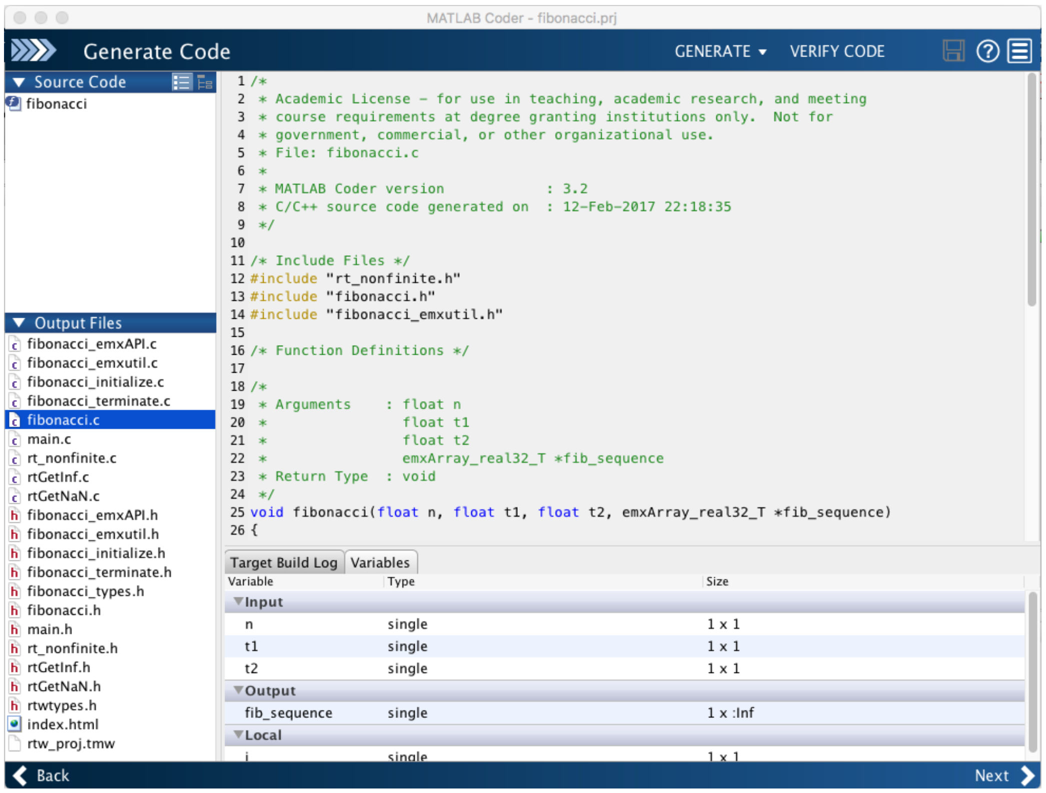
Hardware Board MATLAB Host Computer

Device: Generic (Device vendor), MATLAB Host Computer (Device type)

Toolchain: Automatically locate an installed toolchain

*Fig. 15*

- After the C code is generated, it will display all the .c and .h files associated with the generated C code as shown in Fig. 16. Click Next.
- The next page will display the project summary along with the locations of the generated output.



fibonacci

fibonacci_emxAPI.c
fibonacci_emxutil.c
fibonacci_initialize.c
fibonacci_terminate.c
fibonacci.c
main.c
rt_nonfinite.c
rtGetInf.c
rtGetNaN.c
fibonacci_emxAPI.h
fibonacci_emxutil.h
fibonacci_initialize.h
fibonacci_terminate.h
fibonacci_types.h
fibonacci.h
main.h
rt_nonfinite.h
rtGetInf.h
rtGetNaN.h
rtwtypes.h
index.html
rtw_proj.tmw

```
/*
 * Academic License - for use in teaching, academic research, and meeting
 * course requirements at degree granting institutions only.  Not for
 * government, commercial, or other organizational use.
 * File: fibonacci.c
 *
 * MATLAB Coder version            : 3.2
 * C/C++ source code generated on  : 12-Feb-2017 22:18:35
 */

/* Include Files */
#include "rt_nonfinite.h"
#include "fibonacci.h"
#include "fibonacci_emxutil.h"

/* Function Definitions */

/*
 * Arguments    : float n
 *                float t1
 *                float t2
 *                emxArray_real32_T *fib_sequence
 * Return Type  : void
 */
void fibonacci(float n, float t1, float t2, emxArray_real32_T *fib_sequence)
{
```

Target Build Log | Variables

| Variable | Type | Size |
|---|---|---|
| Input | | |
| n | single | 1 x 1 |
| t1 | single | 1 x 1 |
| t2 | single | 1 x 1 |
| Output | | |
| fib_sequence | single | 1 x :Inf |
| Local | | |
| i | single | 1 x 1 |

*Fig. 16*

## 3.3 Running C Code as iOS App

- In the TestApp project in Xcode, click on the project folder and click on "Add Files to "TestApp". Then, navigate to the folder with the C source code and add all the .c and .h files, as shown in Figs. 17 and 18.
- In the "Algorithm.c" file, add the following preprocessor directives. These directives are used to call the MATLAB Coder functions in the C code.

```
#include "rt_nonfinite.h"
#include "fibonacci.h"
#include "fibonacci_initialize.h"
#include "fibonacci_terminate.h"
#include "fibonacci_emxAPI.h"
```

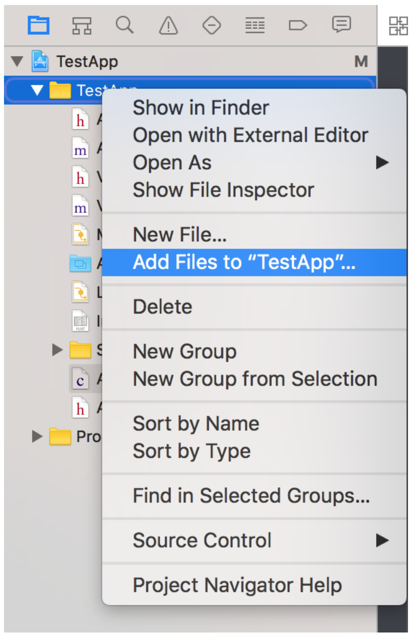

TestApp
M
Show in Finder
Open with External Editor
Open As
Show File Inspector
New File...
Add Files to "TestApp"...
Delete
New Group
New Group from Selection
Sort by Name
Sort by Type
Find in Selected Groups...
Source Control
Project Navigator Help

*Fig. 17*

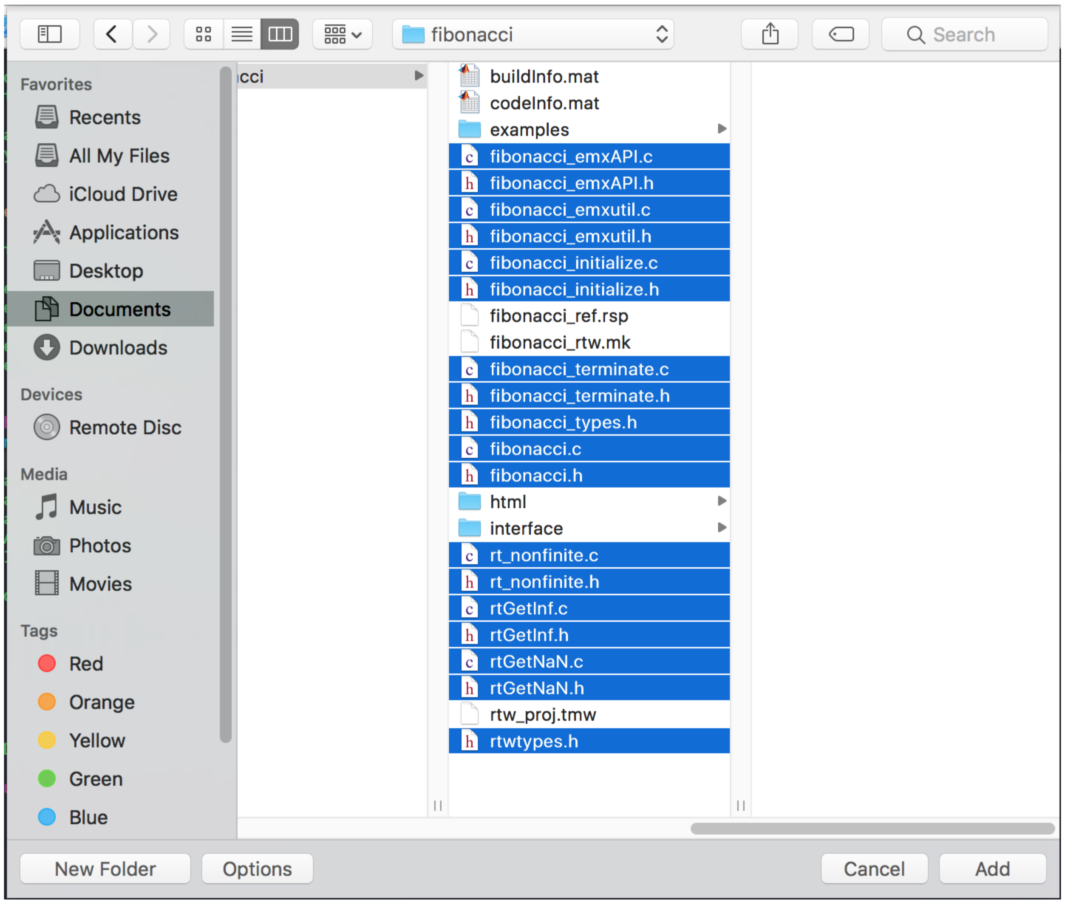
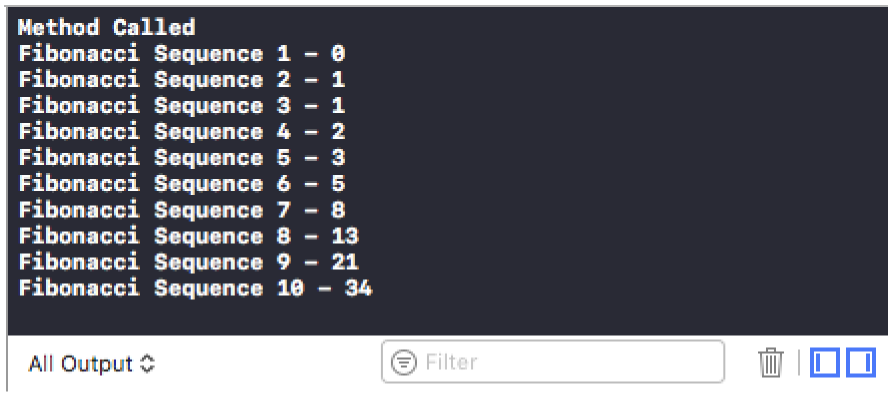

fibonacci
Search
Favorites
Recents
All My Files
iCloud Drive
Applications
Desktop
Documents
Downloads
Devices
Remote Disc
Media
Music
Photos
Movies
Tags
Red
Orange
Yellow
Green
Blue
buildInfo.mat
codeInfo.mat
examples
fibonacci_emxAPI.c
fibonacci_emxAPI.h
fibonacci_emxutil.c
fibonacci_emxutil.h
fibonacci_initialize.c
fibonacci_initialize.h
fibonacci_ref.rsp
fibonacci_rtw.mk
fibonacci_terminate.c
fibonacci_terminate.h
fibonacci_types.h
fibonacci.c
fibonacci.h
html
interface
rt_nonfinite.c
rt_nonfinite.h
rtGetInf.c
rtGetInf.h
rtGetNaN.c
rtGetNaN.h
rtw_proj.tmw
rtwtypes.h
New Folder
Options
Cancel
Add

*Fig. 18*

- In the HelloWorld function in “Algorithm.c”, add the following code before the return function. Since MATLAB has its own data types, this code allows memory to be allocated and then de-allocated when data types have finished their purpose.

```
// Inputs to the MATLAB Function
float n = 10;
float t1 = 0;
float t2 = 1;

// MATLAB array data type definition
emxArray_real32_T *fib_sequence;

// Allocating memory to the MATLAB array
// 2 represents the number of dimensions
emxInitArray_real32_T(&fib_sequence, 2);

// MATLAB function call
// Inputs are entered first in order, followed by the outputs
fibonacci(n, t1, t2, fib_sequence);

// Printing the elements from the Fibonacci Sequence
int i;
for (i = 0; i < n; i++) {
   printf("Fibonacci Sequence %d - %0.0f\n",i + 1, fib_sequence->data[i]);
}

// Deallocate the MATLAB data array
emxDestroyArray_real32_T(fib_sequence);
```

- Run the app. When the button is pressed in the app, the console will be populated with the Fibonacci sequence as shown in Fig. 19.



```
Method Called
Fibonacci Sequence 1 - 0
Fibonacci Sequence 2 - 1
Fibonacci Sequence 3 - 1
Fibonacci Sequence 4 - 2
Fibonacci Sequence 5 - 3
Fibonacci Sequence 6 - 5
Fibonacci Sequence 7 - 8
Fibonacci Sequence 8 - 13
Fibonacci Sequence 9 - 21
Fibonacci Sequence 10 - 34
```

*Fig. 19*

This simple example showcases how to generate a C code from a MATLAB code. More details of the conversion from MATLAB to C codes are provided in the book "Anywhere-Anytime Signals and Systems Laboratory: From MATLAB to Smartphones", which can be acquired from the link:

http://www.morganclaypool.com/doi/abs/10.2200/S00727ED1V01Y201608SPR014

# Section 4: I/O Hardware Dependencies

This section discusses the i/o hardware dependency issues associated with different smartphones or mobile devices and a solution to cope with them.

## 4.1 Using Hardware Preferred Settings

The microphone of a particular smartphone is designed to work with the lowest latency at a certain sampling frequency and with a specific input frame size. To achieve low latency apps, such as noise classification and speech enhancement, the preferred settings by the manufacturer need to be used. Using non-preferred i/o settings increases the i/o latency due to resampling or data rearrangement. Furthermore, care must be taken by not choosing the frame size to be too small as this can lead to frames getting skipped due to a lack of adequate processing time.

The following link provides a listing of optimum frame sizes and sampling frequency for different smartphones and the i/o delays associated with them:

http://superpowered.com/latency

Generally, the preferred sampling frequency for iOS devices is 48 kHz and the minimum input frame size or length is 64 samples.

## 4.2 Maintaining Microphone Consistency

Since different microphones are used in different smartphones, the input frequency characteristics may vary from smartphone to smartphone. For example, some manufacturers favor flat response microphones and some favor microphones with low frequency emphasis for speech processing. Other factors that may affect sound data captured by a smartphone is the location of the smartphone microphone from which audio data are captured, and whether a smartphone cover is placed around the microphone or not.

The difference in microphones and i/o hardware leads to inconsistency among data collected from different smartphones, which would affect the outcome of signal processing algorithms. As a solution, the training and testing of a signal processing algorithm ought to be carried out by the microphone of the same smartphone to decouple the effect of different microphone characteristics and the performance of the algorithm.

## Section 5: Real-Time I/O Implementation

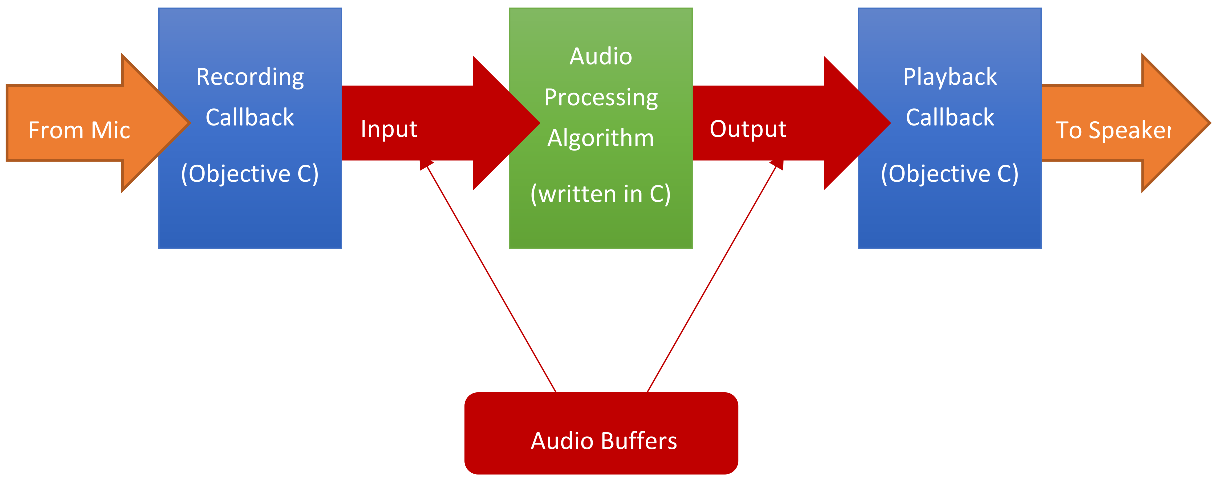

From Mic
Recording Callback (Objective C)
Input
Audio Processing Algorithm (written in C)
Output
Playback Callback (Objective C)
To Speaker
Audio Buffers

*Fig. 20*

To create a real-time audio processing app, the framework shown in Fig. 20 is needed. This framework consists of:

- Recording Callback: This is a synchronous subroutine associated with the smartphone microphone. It is coded in Objective-C. It keeps recording data from the microphone at an i/o hardware allowed sampling frequency and input frame size. Through this subroutine, audio data can be accessed, processed and passed to the smartphone speaker.
- Input Buffer: This is an audio buffer which stores audio data. The data in this buffer can be processed according to a signal processing algorithm for which the app is designed.
- Audio Processing Algorithm: This block is coded in C as the engine of the framework, where data are processed in real-time and then fed to an output to be played by the speaker.
- Output Buffer: This is an audio buffer that collects the processed data from the audio processing algorithm and makes it available to the output.
- Playback Callback: This is a synchronous subroutine associated with the speaker. The audio data from the output buffer are passed to the speaker at the i/o hardware specified sampling rate and frame size.

## 5.1 Creating Audio I/O App

A simple example is provided here to show how an audio i/o app can be created on an iOS device. Follow the steps in section 2.2 to create a new Xcode project. Then, take the following steps:

- In "ViewController.m", import the header files and add the macros and global variables as shown below:

```
#import <AVFoundation/AVFoundation.h>
#import <AudioToolbox/AudioToolbox.h>
#import <AudioUnit/AudioUnit.h>

#define kOutputBus 0
#define kInputBus 1
#define SHORT2FLOAT 1/32768.0
#define FLOAT2SHORT 32768.0;

#define FRAMESIZE 512
#define SAMPLINGFREQUENCY 48000

#ifndef min
#define min( a, b ) ( ((a) < (b)) ? (a) : (b) )
#endif

AudioUnit au;
AudioBuffer tempBuffer;
```

  The header files contain the functions and data types for initializing the hardware settings of the audio app based on the framework indicated in Fig. 19. The macros are global variables which are easily called. For this example, the sampling frequency is set to 48 KHz and the input frame size to 512. AudioUnit allows assigning appropriate properties to the microphone and speaker. AudioBuffer acts as both the input and the output buffer in this example.

- In the "viewDidLoad" method, add the following code:

```
[[AVAudioSession sharedInstance] setCategory: AVAudioSessionCategoryPlayAndRecord error: NULL];
[[AVAudioSession sharedInstance] setMode: AVAudioSessionModeMeasurement error:NULL];
[[AVAudioSession sharedInstance] setPreferredSampleRate:SAMPLINGFREQUENCY error:NULL];
[[AVAudioSession sharedInstance]
setPreferredIOBufferDuration:(float)FRAMESIZE/(float)SAMPLINGFREQUENCY error:NULL];
```

This code sets the category of the audio session to Play and Record. This means that the app will record audio from the microphone and plays it back to the speaker. The mode of the audio session is set to measurement as this mode offers the lowest latency. The preferred sampling rate and i/o buffer duration are also stated as per the macros defined earlier. Note that in case of non-preferred settings, the hardware will set the audio session to the default sampling rate and frame size.

- Audio Component is then set as type output and is assigned to Audio Unit. Also, the input and output properties of Audio Unit are defined.

```
AudioComponentDescription desc;
desc.componentType = kAudioUnitType_Output;
desc.componentSubType = kAudioUnitSubType_RemoteIO;
desc.componentFlags = 0;
desc.componentFlagsMask = 0;
desc.componentManufacturer = kAudioUnitManufacturer_Apple;
AudioComponent component = AudioComponentFindNext(NULL, &desc);
if (AudioComponentInstanceNew(component, &au) != 0) abort();

 UInt32 value = 1;
if (AudioUnitSetProperty(au, kAudioOutputUnitProperty_EnableIO, kAudioUnitScope_Output, 0, &value,
sizeof(value))) abort();
value = 1;
if (AudioUnitSetProperty(au, kAudioOutputUnitProperty_EnableIO, kAudioUnitScope_Input, 1, &value,
sizeof(value))) abort();
```

- In this step, the format of recording audio data is stated. Here, the format is stated as Linear PCM:

```
AudioStreamBasicDescription format;
format.mSampleRate        = 0;
format.mFormatID        = kAudioFormatLinearPCM;
format.mFormatFlags        = kAudioFormatFlagIsSignedInteger;
format.mFramesPerPacket     = 1;
format.mChannelsPerFrame          = 1;
format.mBitsPerChannel              = 16;
format.mBytesPerPacket              = 2;
format.mBytesPerFrame               = 2;
if (AudioUnitSetProperty(au, kAudioUnitProperty_StreamFormat, kAudioUnitScope_Input, 0, &format,
sizeof(format))) abort();
if (AudioUnitSetProperty(au, kAudioUnitProperty_StreamFormat, kAudioUnitScope_Output, 1, &format,
    sizeof(format))) abort();
```

- After Audio Unit is set up, the input and output callbacks to Audio Unit are assigned for capturing and playing audio from the smartphone. Note that this code will throw an error as the callbacks are not defined yet. This error will go away when they get defined.

```
// Set input callback
AURenderCallbackStruct callbackStruct;
callbackStruct.inputProc = recordingCallback;
callbackStruct.inputProcRefCon = (__bridge void *)(self);
AudioUnitSetProperty(au,
            kAudioOutputUnitProperty_SetInputCallback,
            kAudioUnitScope_Global,
            kInputBus,
            &callbackStruct,
            sizeof(callbackStruct));

// Set output callback
callbackStruct.inputProc = playbackCallback;
callbackStruct.inputProcRefCon = (__bridge void *)(self);
AudioUnitSetProperty(au,
            kAudioUnitProperty_SetRenderCallback,
            kAudioUnitScope_Global,
            kOutputBus,
            &callbackStruct,
            sizeof(callbackStruct));
```

- Finally, Audio Buffer and Audio Unit are initialized:

```
tempBuffer.mNumberChannels = 1;
tempBuffer.mDataByteSize = FRAMESIZE * 2;
tempBuffer.mData = malloc( FRAMESIZE * 2 );

AudioUnitInitialize(au);
AudioOutputUnitStart(au);
```

- Next, the callbacks are defined. These are the functions for the audio peripherals that are assigned as pointers and called as synchronous subroutines. In the input callback, audio data are received from the microphone and stored in the temp buffer before getting processed. In the output callback, the processed data are outputted to the speaker.

```
static OSStatus playbackCallback(void *inRefCon,
                                 AudioUnitRenderActionFlags *ioActionFlags,
                                 const AudioTimeStamp *inTimeStamp,
                                 UInt32 inBusNumber,
                                 UInt32 inNumberFrames,
                                 AudioBufferList *ioData) {

    for (int i=0; i < ioData->mNumberBuffers; i++) {
        AudioBuffer buffer = ioData->mBuffers[i];
        UInt32 size = min(buffer.mDataByteSize, tempBuffer.mDataByteSize);
        memcpy(buffer.mData, tempBuffer.mData, size);
        buffer.mDataByteSize = size;
    }
    return noErr;
}

static OSStatus recordingCallback(void *inRefCon,
                                  AudioUnitRenderActionFlags *ioActionFlags,
                                  const AudioTimeStamp *inTimeStamp,
                                  UInt32 inBusNumber,
                                  UInt32 inNumberFrames,
                                  AudioBufferList *ioData) {

    AudioBuffer buffer;
    ViewController* view = (__bridge ViewController *)(inRefCon);

    buffer.mNumberChannels = 1;
    buffer.mDataByteSize = inNumberFrames * 2;
    buffer.mData = malloc( inNumberFrames * 2 );

    // Put buffer in a AudioBufferList
    AudioBufferList bufferList;
    bufferList.mNumberBuffers = 1;
    bufferList.mBuffers[0] = buffer;

    AudioUnitRender(au,
                    ioActionFlags,
                    inTimeStamp,
                    inBusNumber,
                    inNumberFrames,
                    &bufferList);

    [view processAudio:&bufferList];



    return noErr;
}
```

The processAudio method will throw an error as it has not been defined yet. This method is used to process the data stored in the audio buffer. A blank method is created here for now.

```
    - (void) processAudio: (AudioBufferList*) bufferList{
    }
```

- Next, create a new .c and .h file similar to section 2.4 and name it FIR. In FIR.c, add this simple moving average filtering code:

```
    void FIR(float* input, float* output, int nSamples) {
        int i = 0;

        static float endSamples[2] = {0,0};

        for (i = nSamples - 1; i > 1; i--) {
            output[i] = (input[i] + input[i - 1] + input[i - 2])/3;
        }

        output[1] = (input[1] + input[0] + endSamples[1])/3;
        output[0] = (input[0] + endSamples[1] + endSamples[0])/3;

        endSamples[1] = input[nSamples - 1];
        endSamples[0] = input[nSamples - 2];


    }
```

And in FIR.h, declare this function:

```
    void FIR(float* input, float* output, int nSamples);
```

- After the C code is created, in View Controller.m, import the FIR.h file and in the “processAudio” method, add the following code:

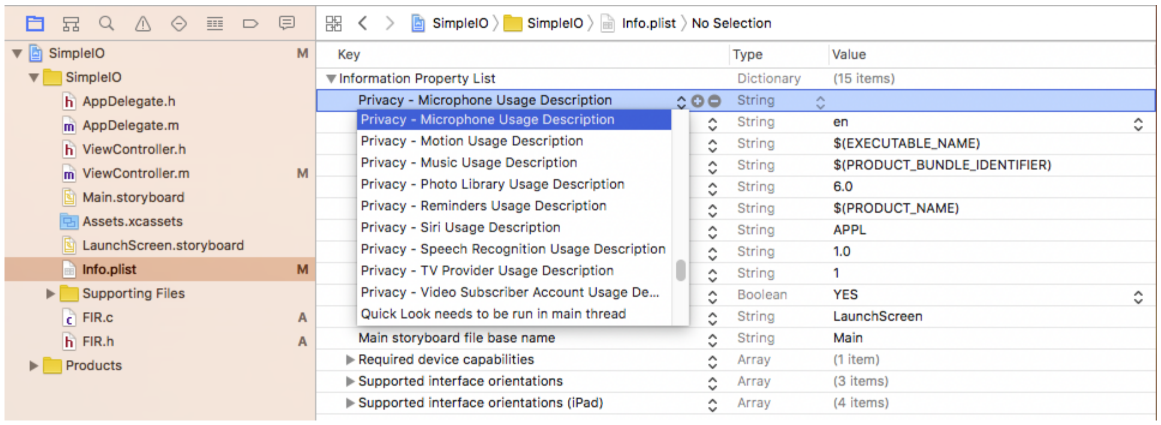
```
    AudioBuffer sourceBuffer = bufferList->mBuffers[0];
```

```
short *buffer = (short*)malloc(sourceBuffer.mDataByteSize), *output =
(short*)malloc(sourceBuffer.mDataByteSize);
float *input = (float*)malloc(sizeof(float*) * 512), *float_output = (float*)malloc(sizeof(float*) * 512);


memcpy(buffer, bufferList->mBuffers[0].mData, bufferList->mBuffers[0].mDataByteSize);



for (int i = 0; i < 512; i++) {
   input[i] = buffer[i] * SHORT2FLOAT;
}

FIR(input, float_output, FRAMESIZE);

for (int i = 0; i < 512; i++) {
   output[i] = float_output[i] * FLOAT2SHORT;
}

if (tempBuffer.mDataByteSize != sourceBuffer.mDataByteSize) {
   free(tempBuffer.mData);
   tempBuffer.mDataByteSize = sourceBuffer.mDataByteSize;
tempBuffer.mData = malloc(sourceBuffer.mDataByteSize);
}

memcpy(tempBuffer.mData, output, bufferList->mBuffers[0].mDataByteSize);
free(buffer);
free(output);
free(float_output);
free(input);
```

In the above code,

- the audio in “tempBuffer” is stored in a buffer
- converted to float
- processed via the FIR function
- stored in an output buffer
- converted to short
- then stored again in “tempBuffer”

- Finally, the app can be made to ask for access to the microphone. This can be done by adding the property as illustrated below in Fig. 21 in the Info.plist file in the navigator window:



SimpleIO 〉 SimpleIO 〉 Info.plist 〉 No Selection

```
▼ SimpleIO                     M
  ▼ SimpleIO
      AppDelegate.h
      AppDelegate.m
      ViewController.h
      ViewController.m         M
      Main.storyboard
      Assets.xcassets
      LaunchScreen.storyboard
      Info.plist               M
    ▶ Supporting Files
      FIR.c                    A
      FIR.h                    A
  ▶ Products
```

| Key | Type | Value |
|---|---|---|
| ▼ Information Property List | Dictionary | (15 items) |
| Privacy - Microphone Usage Description | String | |
| | String | en |
| | String | $(EXECUTABLE_NAME) |
| | String | $(PRODUCT_BUNDLE_IDENTIFIER) |
| | String | 6.0 |
| | String | $(PRODUCT_NAME) |
| | String | APPL |
| | String | 1.0 |
| | String | 1 |
| | Boolean | YES |
| | String | LaunchScreen |
| Main storyboard file base name | String | Main |
| ▶ Required device capabilities | Array | (1 item) |
| ▶ Supported interface orientations | Array | (3 items) |
| ▶ Supported interface orientations (iPad) | Array | (4 items) |

Privacy - Microphone Usage Description
Privacy - Motion Usage Description
Privacy - Music Usage Description
Privacy - Photo Library Usage Description
Privacy - Reminders Usage Description
Privacy - Siri Usage Description
Privacy - Speech Recognition Usage Description
Privacy - TV Provider Usage Description
Privacy - Video Subscriber Account Usage De...
Quick Look needs to be run in main thread

*Fig. 21*

The code described above can be run as an app. This app will not run on the simulator properly and needs to be run on an actual smartphone. Note that what has been described here is just one way of creating an i/o for an iOS device. For more details on audio units, the interested reader is referred to the following link:

https://developer.apple.com/library/content/documentation/General/Conceptual/ExtensibilityPG/AudioUnit.html#//apple_ref/doc/uid/TP40014214-CH22-SW1

Also, it is worth mentioning that the following third-party tools are quite helpful in creating audio apps:

- http://superpowered.com
- http://audiokit.io