**System Design**

**Simplified Digital Masker**

**Program Architecture**

**Prepared By**

**R. S. Hentz**

**12/13/2022**

**Version 1.0**

**Version History**

|  |  |  |
| --- | --- | --- |
| **Version** | **Date** | **Description/Notes** |
| **1** | 12/13/2022 | Initial Release |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

**Table of Contents**

[Abstract 4](#_Toc120869957)

[Introduction 4](#_Toc120869958)

[Purpose 4](#_Toc120869959)

[Design Requirements 4](#_Toc120869960)

[System Design Concept 5](#_Toc120869961)

[Background 5](#_Toc120869962)

[ESP32 Features 5](#_Toc120869963)

[RTOS Capabilities 6](#_Toc120869964)

[Defining Dominant Frequency 7](#_Toc120869965)

[Implementation 10](#_Toc120869966)

[Original Masker Algorithm Implementation 10](#_Toc120869967)

[Original Masker Considerations 10](#_Toc120869968)

[Frequency Doubler Algorithm Implementation 11](#_Toc120869969)

[Frequency Doubler Algorithm Considerations 11](#_Toc120869970)

[Delay Masker Algorithm Implementation 11](#_Toc120869971)

[Delay Masker Algorithm Considerations 11](#_Toc120869972)

[Conclusion 11](#_Toc120869973)

[References 11](#_Toc120869974)

# Abstract

The Messiah University Collaboratory Frequency Altered Device (FAD) team is undertaking the task of building a digital device to replace the Edinburg Masker. As part of this effort, the Digital Masker Design Team is imp[lamenting an improved masker offering three different masking algorithms using a microcomputer as the primary electronic component of the solution. This report presents a simple design solution employing an easily implemented time domain analysis of the audio input signal in order to define and produce the desired audio output required of the masker.

# Introduction

The Messiah Collab Digital Masker Design team is in the process of implementing a digital version of the Edinburg Masker. The team has elected to implement this digital masker using the Expessif ESP 32 microprocessor chip running the ESP-32 RTOS operating system. Using this microprocessor, the team wants to implement three different masker algorithms to include:

1. Delay Masker Algorithm – produces a time delayed output signal which closely tracks the audio input signal but which is delayed by a fixed timeframe ranging from 20 to 100 milliseconds.
2. The Original Masker Algorithm – produces a simulated white noise output by producing a saw-toothed wave form at a frequency near the dominant frequency of the audio input signal. This output closely resembles the output produced by the original masker electronics.
3. Frequency Doubler Algorithm – produces an output signal at 2 times the dominant frequency of the input signal.

# Purpose

The purpose of this System Design Document is to define an program implementation architecture which allows for implementing the three desired algorithms utilizing the features of the ESP-32 and FreeRTOS operating system. This architecture will employ standard FreeRTOS and FreeRTOS library capabilities while simplifying the program design steps required to implement the three algorithms. Specifically, this architecture will avoid the use of any complex mathematical libraries to process signal characteristics and or compute signal frequency.

# Design Requirements

The following critical characteristics will drive the design of the program architecture:

1. The Audio Input signal at the input to the ESP-32 shall be a signal of amplitude 1.5 v +- 1.5 v at a frequency in the range of 50 Hz to 500 Hz.
2. The Audio output signal shall be a signal of amplitude 1.5 v +- 1.5 v at a frequency in the range of 50 Hz to 1000 Hz.
3. The algorithm selection shall be by means of a pushbutton which when depressed will cycle from one algorithm to the next in a continuous loop.

# System Design Concept

Figure 1 ‘ESP Concept Design,’ provides a high-level overview of the digital masker implementation concept.

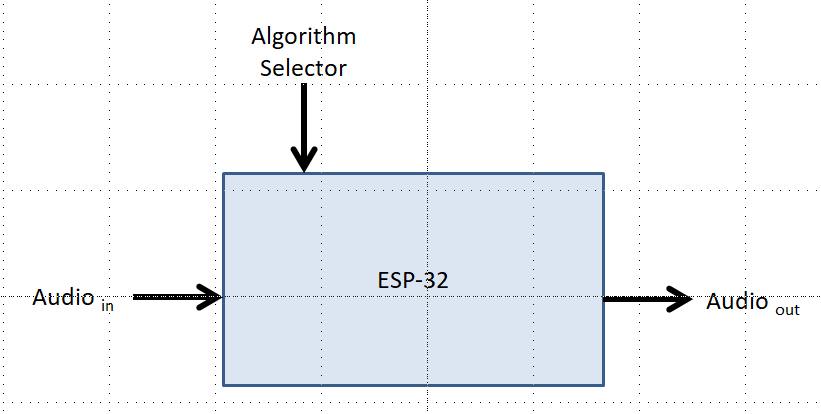


Figure ESP Concept Design

As illustrated in Figure 1, the audio input signal is received by the microprocessor, and depending on the algorithm selected, the device generates the appropriate audio output signal.

The signal transformation produced by each of the chosen algorithms is discussed later in this document.

# Background

In order to fully explain the proposed solution it is necessary to have a basic understanding of the capabilities of the ESP-32, the FreeRTOS operating system and the selected approach to computing the predominant frequency of the input audio signal. The following sections provide this necessary information, so that the solution approach can be fully understood.

## ESP32 Features

The ESP32 is a low-cost, low-power microcontroller, manufactured by Espressif Systems, with an integrated Wi-Fi and dual-mode Bluetooth. The ESP32 series can employ either a single-core or dual-core configuration running at either 160 MHz or 240 MHz depending on the model selected. Figure 2, ‘ESP32 Functional Block diagram’ illustrates the functionality available with this board. [REF #1} Of key importance to this design effort is the availability of two 8 bit Digital to Analog Converters (DAC); up to 18 channels of 12 bit Analog to Digital Converters (ADC); and the Real Time Clock (RTC) functionality. Additionally, for future upgradability, the Bluetooth interface is also necessary when the design advances to utilizing this interface rather than hardwired microphone and headphones.

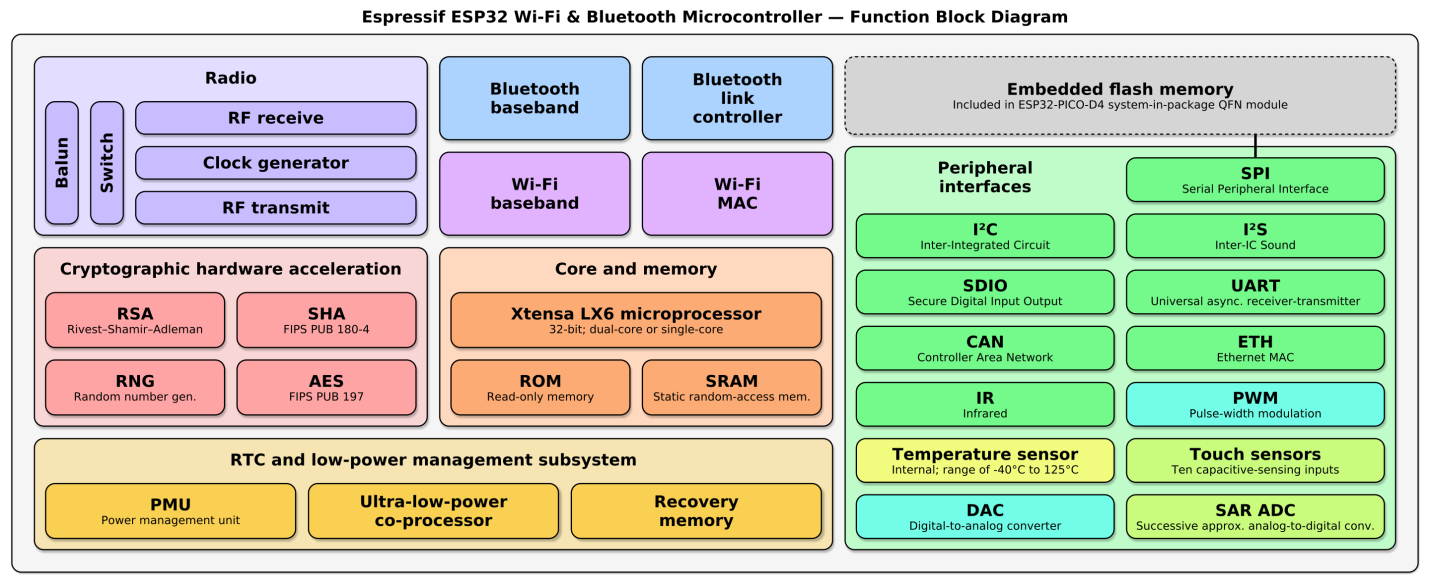


Figure ESP32 Functional Block Diagram

## RTOS Capabilities

The ESP32 port of FreeRTOS has been selected to run on the microprocessor for purposes of this development activity. FreeRTOS was selected because of the following basic features and capabilities [REF # 3]:

* Is known to be reliable and feature rich.
* Has a minimal ROM, RAM and processing overhead. Typically an RTOS kernel binary image will be in the region of 6K to 12K bytes.
* Is free for use in commercial applications.
* Has an active, monitored support forum.
* Provides ample documentation.
* Provides significant library support to simplify implementation of interfaces and signal processing.
* Is scalable, simple and easy to use.
* Allows for real time processing of events.

## Defining Dominant Frequency

The determination of the dominant input frequency is a key characteristic of both the Original Masker Algorithm and the Frequency Doubler Algorithm. Two options exist for computing average frequency:

* Convert the time domain signal received by the microcomputer into a frequency domain using a function such as a Fast Fourier Transform (FFT). Once the frequency components of the signal are isolated, the microcomputer would then ascertain that primary frequency from the array produced by the FFT. The use of the FFT function was felt to be too complex, computationally intensive and would require too much time to compute for accurately tracking an audio input signal in realtime.
* Utilize the time domain characteristics of the incoming signal to measure the time period of several input cycles and average the results over several cycles to arrive at an average input signal period.

Given the objective of this design document, the second approach of utilizing the time domain properties of the audio input signal was selected as the approach to be pursued. This methodology employs the basic FreeRTOS architecture to create two tasks.

* The first task is simply a timer task which controls the sampling rate at which the input signal is measured; the output signal is changed; and rate at which the second task is executed.
* The second task processes the input signal level measured by the first task and computes the amplitude of the output signal to be used, the next time the first task runs. Note: The computation of the output signal amplitude is dependent on the masker algorithm selected.

This basic program flow is illustrated in Figure 3, “High level Process Flow.”

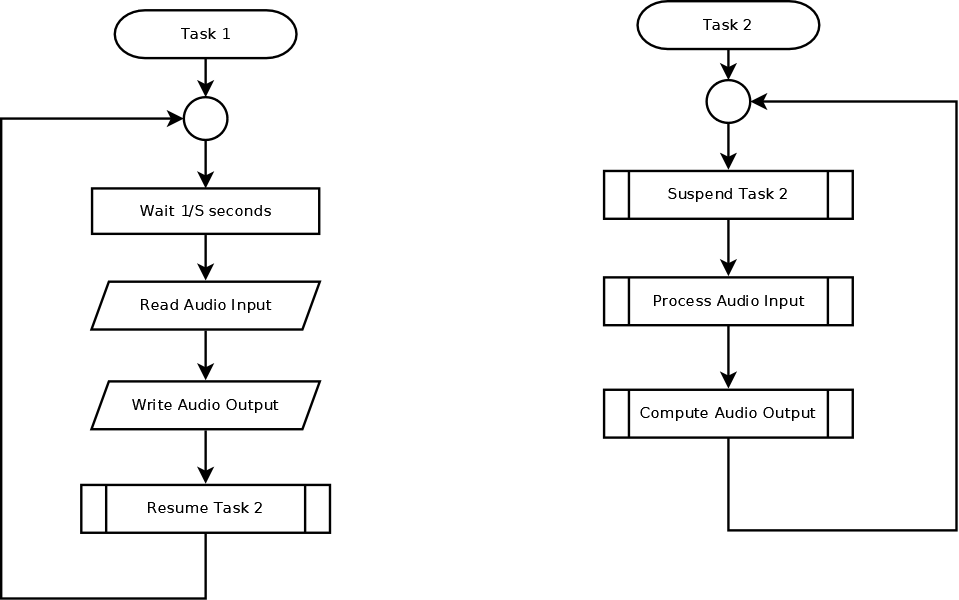


Figure High level Process Flow

The success of this simplified approach to development of the digital masker is the ability to determine the predominant frequency of the input signal using information available solely in the time domain. To demonstrate how this can be accomplished, several controlling variables are defined as listed below:

* ***R*** = Rolling Average - the number of signal input half periods used to compute the average period of the input signal. Typically, this value is in the range of 5 to25. The smaller number provides better responsiveness to signal frequency change at a loss of frequency estimating accuracy.
* ***S*** = Sample rate in samples/sec at which the input signal is sampled. Typically, this rate is set at something greater than 41Khz for quality digitization of audio data.
* ***THR*** = TheAmplitude threshold denoting the input signal crossover value. Note: This value is typically determined by ½ the maximum input of the ADC used to measure the input signal.
* **AI*x*** = Amplitude of Input Signal at sample time *x* as measured by theADC.

In order to simplify the computation, EQ #1 declares a boolean variable THX indicative of the amplitude of the input signal relative to the control variable THR.

|  |  |
| --- | --- |
|  | EQ #1 |
| Where:  TX*x* = Boolean State indicating that amplitude of Audio Input signal is above the value of THR.  THR = TheAmplitude threshold denoting the input signal crossover value. |  |

Using EQ#2 the half period of the Audio Input Signal is computed.

|  |  |
| --- | --- |
|  | EQ #2 |
| Where:  ISC*x* = Sample count of input signal period over the current Half cycle at sample time x. |  |

In EQ#3 the value of ISC*x* is used to compute the Average Input Signal Half Period, considered for purposes of this computation as the dominate half period. From the dominate half period, the Dominate Frequency of the Audio Input signal is also defined.

|  |  |
| --- | --- |
|  | EQ #3 |
| Where:  = Average Input Signal Half Period count over last R period changes at time *x*. This is considered the dominate half period of the input signal.  = Dominate Frequency at time *x*. |  |

# Implementation

In this section, a high level analysis of the methods required to implement the three masker algorithms are presented given the use of the two task approach presented in the high level process flow discussion.

## Original Masker Algorithm Implementation

The original masker is an analog device which employs a Phase Lock Loop (PLL) to create a voltage proportional to the frequency of the audio input. The PLL output then fed a Voltage Controlled Oscillator which generates a Saw-toothed output signal at the dominant frequency of the audio input. To replicate the signal output for the original masker the digital masker will implement the algorithm defined in EQ #4

|  |  |
| --- | --- |
|  | EQ #4 |
| Where:  = The dominate half period of the input signal at sample time x.  OSC*x* = The Output Signal Sample Count at sample time x.  AO*x* = The amplitude of the Output Signal at sample time x |  |

## The effects of implementing the Original Masker Algorithm when provided with a constant frequency input signal are illustrated using a python implementation in Figure 4.

## 

Figure Original Masker Algorithm Results

## Original Masker Considerations

There are three key considerations when implementing this algorithm.

1. The amplitude Audio Output Signal will always range from 0 v to the maximum voltage delivered by the ESP-32 DAC. The maximum DAC output is dependent on the power supply used to power the microcomputer chip.
2. The setting for THR, in this implementation is a fixed value, selected to be at a value consistent with 1/2 the ESP-32’s ADC output at maximum Audio Input Signal.
3. The maximum Audio Input signal is considered a constant value.

## Frequency Doubler Masker Algorithm Implementation

The Frequency Doubler Masker Algorithm masks the speakers voice by doubling the frequency of the audio input signal and outputting the resultant signal to the speaker’s ear piece. This algorithm is implemented using the mathematical operations shown in EQ #5

|  |  |
| --- | --- |
|  | EQ #5 |
| Where:  = The dominate half period of the input signal at sample time x.  OSC*x* = The Output Signal Sample Count at sample time x.  AO*x* = The amplitude of the Output Signal at sample time x  DACm = Maximum Output value of the ESP DAC |  |

## The effects of implementing the Frequency Doubler Masker Algorithm when provided with a constant frequency input signal using a python implementation are illustrated in Figure 5.

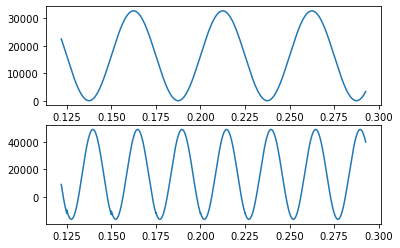


Figure Doubler Masker Algorithm Results

## Frequency Doubler Masker Algorithm Considerations

The following considerations should be taken into account when implementing this algorithm.

1. The output signal frequency will be limited by the response capability of the audio receptor used to convert audio energy to physical vibrations.
2. The implementation assumes the audio\_out signal is produced at a constant volume defined by the maximum output level of the ESP DAC.

## Delay Masker Algorithm Implementation

The Delay Masker Algorithm masks the speakers voice by delaying the audio input signal and outputting the resultant signal to the speaker’s ear piece. The delay alters the audio sound and masks the speakers voice. In order to implement the delay algorithm, it is necessary to introduce a FIFO Message Queue of a length equal to the number of samples executed in the delay time period. The use of the message queue, allows the audio\_input process to load data into the queue, while the audio\_output process independently retrieves data from the queue. Figure 6, shows conceptually how these processes utilize the message queue.

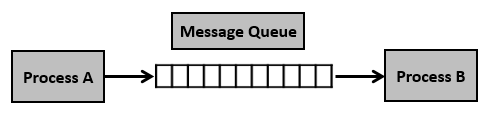


Figure Conceptual Operation of Message Queue

With the Message Queue implemented, the algorithm is uses the mathematical operations shown in EQ #6 to create the desired output.

|  |  |
| --- | --- |
|  | EQ #6 |
| Where:  D = the desired output delay time in seconds.  dt = the delay in terms of sample offset.  ISCt = the total number of input samples incurred at time t  OSC*x* = the Output Signal Sample Count at sample time x.  Idx = the buffer position for the audio input signal at time x.  Aix = the audio\_input signal read at time x  AO*x* = The amplitude of the Output Signal at sample time x |  |

## The effects of implementing the Delay Masker Algorithm when provided with a constant frequency input signal using a python implementation are illustrated in Figure 7.

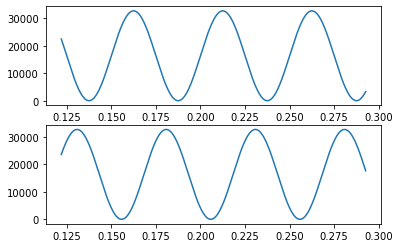


Figure Delay Masker Algorithm Results

## Delay Masker Algorithm Considerations

Implementation of this solution is dependent on the following considerations.

1. The desired delay is a constant and is expressed in terms of partial seconds of offset.
2. The implementation assumes the audio\_out signal is produced at a constant volume defined by the maximum output level of the ESP DAC.

# Conclusion

The ability to utilize a simple time domain analysis of audio input signal has been shown to provide for adequate audio\_output signal performance of the three desired digital masker algorithms using a simple python implementation. In this implementation, the most complex mathematical calculation utilized the standard built-in math capability present both in the python and in C standard implementations. The next step will be to implement these concepts in C for execution within FreeRTOS on the ESP platform

# References

The following references have been used to develop this document and the related design.

1. [ESP32](https://en.wikipedia.org/wiki/ESP32).
2. [FreeRTOS - ESP32 - — ESP-IDF Programming Guide](https://docs.espressif.com/projects/esp-idf/en/latest/esp32/get-started/index.html).
3. [Why freeRTOS?](https://icircuit.net/esp32-introduction-freertos/1297)