Design Brief: Group 5

Gabriel Apap, Damjan Filipovic, Mark Mizzi

March 13, 2023, Document v.6700

This document outlines the design of a DTMF encoding system based on a microcontroller board. The system described makes use of an LCD display to show the user what keys they have pressed as well as to display any messages or errors. An LED is used as an indicator of the system's state. An on-board DAC, in conjunction with an amplifier circuit and speaker are used to generate and amplify the DTMF tones.

1 Introduction

Dual Tone Multi Frequency (DTMF for short) encoding is a technology used to communicate with other devices over a standard analogue telephone line [1]. This is mainly used for automated switchboards for calls; however, this is also used in remote control systems, telephone banking, and other applications [1].

The technology functions by converting each of the 16 symbols on a keypad (1-9, A-D, * and #) into a specific tone. The tone consists of a sum of two sine waves, whose frequency is determined by pairwise combination of 4 high frequencies (representing the keypad columns) and 4 low frequencies (representing the keypad rows) [1].

The goal of this project is to implement a DTMF encoding system based on the Embedded Artists LPC4088 microcontroller board.

Microcontrollers are designed for use in real-time systems, and hence do not have an operating system which deals with the issue of task management. Task scheduling has to be handled explicitly by the programmer, through the use of an ad-hoc scheduler and/or interrupt handlers.

In addition, microcontrollers are generally low-cost systems with limited hardware resources, and hence require the use of low-level programming tricks to achieve efficiency and performance despite the resource constraints.

The DTMF system implemented will make use of several I/O devices, including an LCD used to convey user input or settings options to the user, an indicator LED which turns on when the system is booted, and a keypad input device used to choose between settings or to input DTMF symbols.

On boot-up, the system will prompt the user to enter one of two operational modes: settings or normal mode.

In settings mode, the user is presented with numbered options which can be used to configure the system. The user's options are persisted to flash memory for use in normal mode. The options available to the user will include:

1. Inter-symbol spacing, i.e. the delay between two consecutive tones being produced.
2. Symbol length, i.e. the duration for which a single tone is produced.
3. Tone resolution. This setting affects the sampling frequency of the tone produced.
4. Reboot system. This saves the chosen options and resets the system.

When the system enters normal mode, options are loaded from flash memory, and the system behaves as a DTMF encoder, producing tones when the user presses keys.

Section 2 describes the proposed design for the DTMF encoding system. The system's architecture is described in subsection 2.1. Subsection 2.2 clarifies some implementation details regarding the keypad input.

Section 3 outlines a breakdown of the system's implementation into individual tasks, as well as a schedule for implementing these tasks.

2 System Design

2.1 System architecture

To implement these features while adhering to good seperation of concerns, the system must make use of several software components. The proposed

software components are shown in Figure 1.

Not shown in this figure is the use of several global variables to store the state of the system. These include (i) a thread-safe flag which indicates whether a tone is being generated, (ii) a counter used to keep track of tone generation in between interrupt handler invocations, (iii) a read-only lookup-table containing samples of the sin function, and (iii) a `struct` holding user configuration of the system.

It is also worth noting that the lock-free queue indicated in the design is globally allocated, and is accessed by several "threads" of execution.

The responsibilities and features of each of the software components indicated in the figure will now be described.

The **mode loader** is the entry point of the system. It turns on the indicator LED, and presents the user with options for choosing between settings mode or normal mode. It must also process any key press which indicates the user's choice, and load the code for settings or normal mode (through a function call).

The **settings** component will contain the code for settings mode. It must present the user with options for configuring the system, and process the user's choice. When the user chooses to re-boot the system, it must also persist the chosen configuration to flash memory.

The **DAC driver** is central to system operation, as it contains functions which generate the DTMF tones. The module should provide a function which, given a symbol, enables a timer interrupt handler (depending on what tone is to be produced) and sets the timer duration to the sampling frequency of the tone.

The module should also implement 16 timer interrupt handlers (one for each tone) to produce tone samples and drive the DAC. The interrupt handlers will be similar in behaviour, and will share a lot of their code. The benefits of having an interrupt handler for each tone is that it reduces the amount of global state used by the system and allows the compiler to aggressively optimize the handlers in question, e.g. applying constant division optimization.

The behaviour of each interrupt handler is shown in the control flow diagram in Figure 2.

The **keypad driver** component provides an abstraction layer over the keypad. It must implement a single polling cycle (as a function) for use in other modules.

In addition, it must implement a method that whilst running the polling cycle, performs the following actions immediately on detection of a key press:

1. It immediately outputs the DTMF symbol detected to the LCD screen.
2. It checks if there is a tone interrupt handler enabled. This can be done by atomic test and set on the global flag.
3. If there is no tone interrupt handler enabled, it takes the symbol detected, and enables a tone-generating interrupt handler to generate its tone (using a method in the DAC driver).
4. If there is already a tone interrupt handler enabled, it pushes the DTMF symbol onto the output queue.

This method will be used in normal mode to process key presses.

The **persistent storage driver** provides an abstraction layer over the flash memory. It provides functions to serialize and deserialize the C structs which need to be stored into a byte stream, and allows for loading or storing serialized data from/to flash memory. The use of a wear leveling algorithm when storing data is essential to prevent the flash memory from degrading quickly[2].

The **LED driver** provides methods to turn the indicator LED on or off.

The **LCD driver** provides an abstraction layer over the LCD screen. It must implement methods which allow a DTMF symbol or a C string to be encoded and displayed on the LCD.

The **lock-free queue** is a globally allocated thread-safe data structure. It needs to support the following thread-safe (atomic) operations:

1. Enqueue a DTMF symbol.
2. Check if queue is empty and dequeue. This must be one atomic operation.

2.2 How the polling method is invoked

There are several ways in which the polling cycle procedure that processes user input in normal mode (described above) can be executed.

The simplest method is to run the procedure in application mode, with a set period of busy waiting in between invocations of the method to avoid spurious input. (A human user will keep a key pressed for several clock cycles.)

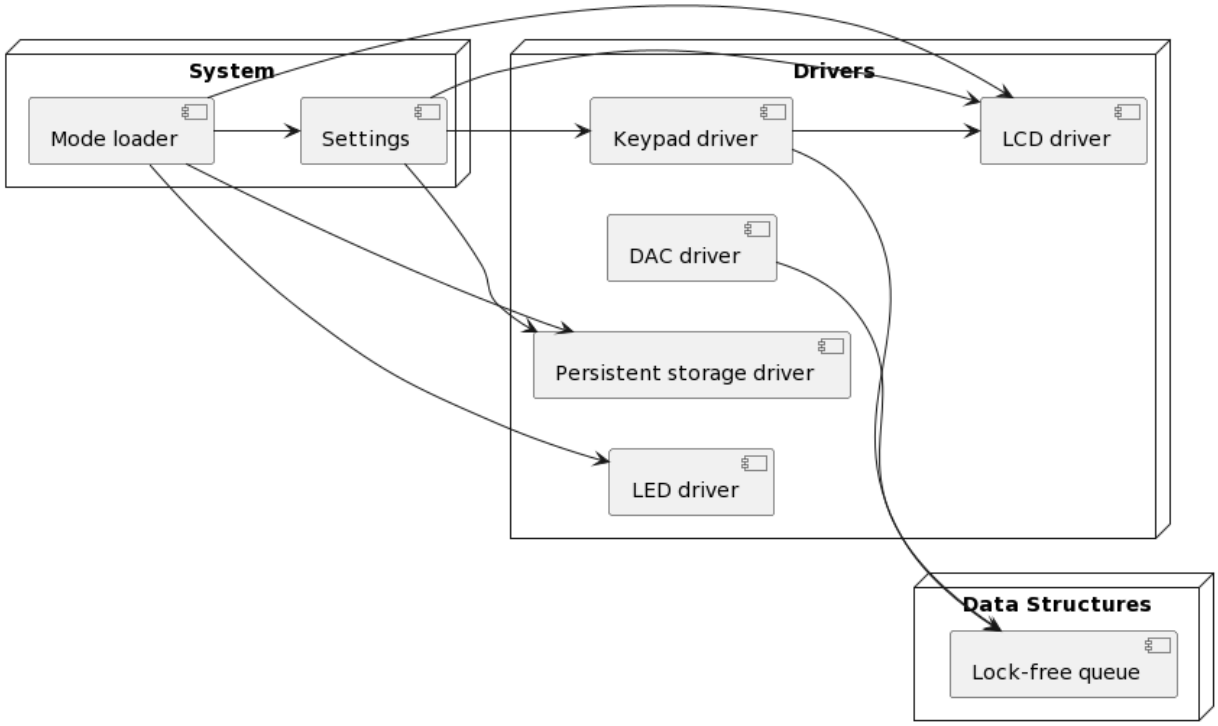

Figure 1: Software components proposed for the system, grouped by functionality. Arrows between components indicate caller-callee dependencies.

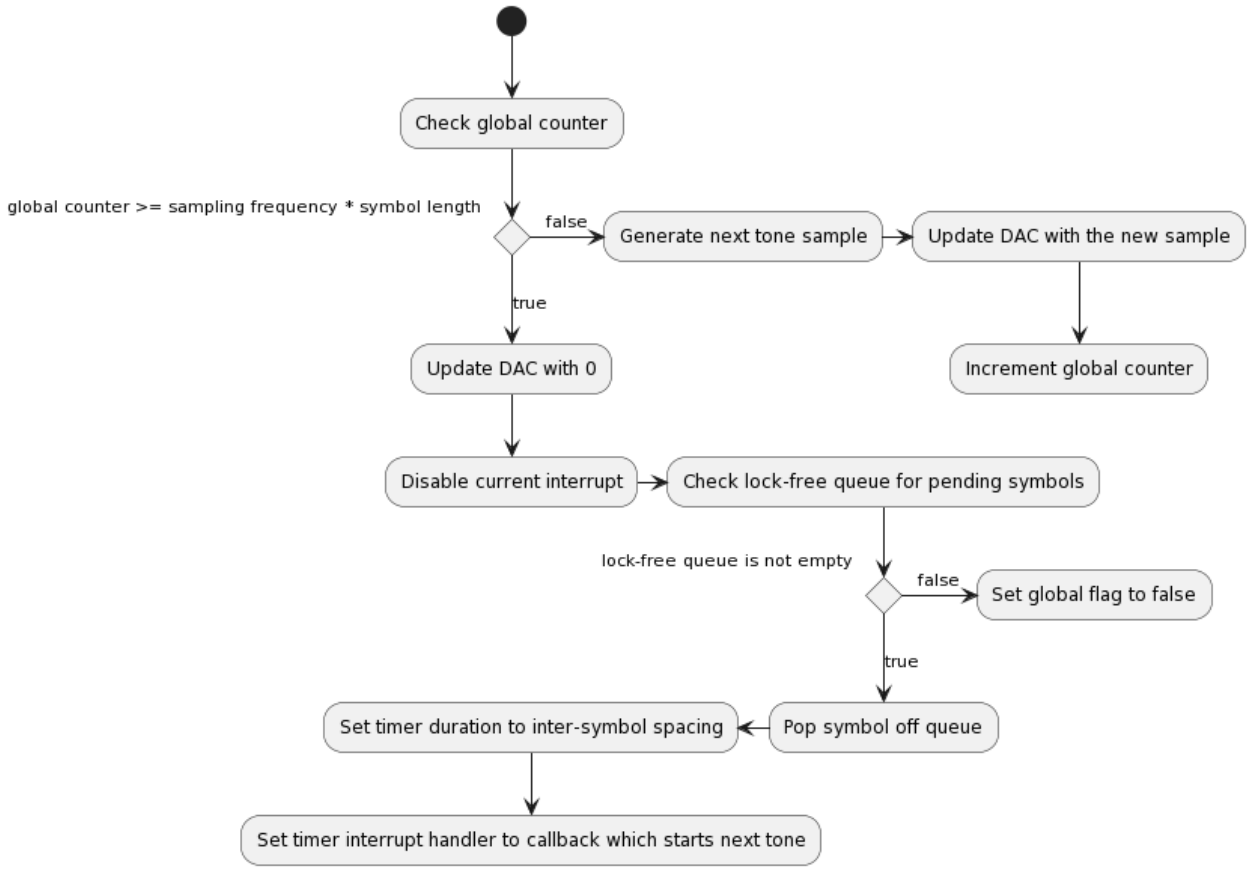

Figure 2: Control flow diagram illustrating behaviour of tone-generating interrupt handlers

A more sophisticated option is to run the procedure as an interrupt handler for a timer interrupt, with the timer duration set to avoid spurious input. In this version, the processor has no application code to run and can be put in sleep mode in between interrupt invocations. While more efficient than the busy-waiting approach, this method can still invoke the interrupt handler when there is no input to process.

A final, more efficient approach is to have a keypad-driven interrupt handler. In this approach, the columns of the keypad are set high, and pressing any key triggers a pin interrupt for its row. The interrupt handler then invokes the polling method to find which key in that row was pressed and handle it.

Each of these design choices is more sophisticated than the last, while re-using much of the same code. The plan is to implement each of these designs in turn. When one implementation has been successfully tested, the team will move on to implementing a more sophisticated design.

3 Management

3.1 Final Deadline

The final deadline will be set to **10th May** to ensure that the work is done at a proper pace whilst leaving enough room for any necessary adjustments.

3.2 Tasks

To implement the system, the above-mentioned software components have to be split into multiple sub-tasks. Each task will have a fairly allocated time-frame for completion. The deadlines for each task are as follows:

3.2.1 Drivers

1. LCD Driver - Deadline: **8th April**. Subtasks: Symbol and C-String to LCD function, Display function.
2. Keypad Driver - Deadline: **8th April**. Subtasks: Polling Cycle function for the keypad, Keypad driven interrupt.
3. LED Driver - Deadline: **8th April**. Subtasks: LED On function, LED Off function.
4. Persistent Storage Driver - Deadline: **16th April**. Subtasks: Serialising Data, Persisting Data, Loading Data, Deserializing Data.
5. DAC Driver - Deadline: **20th April**. Subtasks: Timer interrupt handler enable method, Set timer duration function, Interrupt handlers.

3.2.2 Data Structures

1. Lock-Free queue - Deadline: **1st April**. Subtasks: Atomic Enqueue function, Atomic Check and Dequeue function.

3.2.3 System

1. Mode loader - Deadline: **16th April**. Subtasks: Display Options function, Load settings/normal mode functions.
2. Settings Mode - Deadline: **27th April**. Subtasks: Display settings function, Save settings function.

3.3 Task Dependencies

When considering the inter-dependencies of the tasks listed above, there is a discrete order of task development that when followed should ensure that each subsequent task has adequate resources to be accomplished. The order is the following:

1. Lock-Free Queue
2. LCD Driver
3. LED Driver
4. Keypad Driver
5. Persistent Storage Driver
6. DAC Driver

4 Closure

In conclusion, this document describes the design of an efficient and performant system implementing DTMF encoding. The system's software has been split up into several modules showing good separation of concerns. Although not discussed here, a highly efficient algorithm has been derived for use in tone generation.

References

[1] Silicon Labs, *DTMF Decoder Reference Design*, 5 2012. Rev. 0.11.

[2] A. Silberschatz, G. Gagne, and P. Galvin, "Operating system concepts," ch. 11, Wiley, 2018.