Design Brief: Group 1

Kevin Costner, Sean Connery, Charles Martin Smith, Andy Garcia, Richard Bradford

February 22, 2023, Document v.6594

Write an executive summary for your project. This should not overflow into the next page, should not contain references, and should be readable by a wide audience.

1 Introduction

Write an introduction, explaining the purpose and background of this project. Give a brief description of the game to be implemented. References to be cited like so [1]. Also include an overview of this document.

2 System Design

2.1 System overview

The system will have two modes of operation, a mode for settings and

2.2 Software Modules

The software modules proposed for the design are shown in Figure 1.

Their responsibilities are as follows:

1. Mode loader
   a) Turn on the indicator LED (through the use of the LED driver).
   b) Present the user with options for choosing between settings mode, or normal mode (through the LCD driver).
   c) Process user's input (through the use of the button matrix driver).
   d) Load the system configuration from persistent storage if the normal mode is chosen (through the use of the persistent storage driver).
   e) Load code for the settings/normal mode depending on the user's choice.
2. Settings
   a) Present the user with options for configuring the system (through the use of the LCD driver).
   b) Process user's input (through the use of button matrix driver).
   c) Persist the system configuration (through the use of the persistent storage driver).
3. Button matrix driver The behaviour of this component partially depends on the system's mode of operation.
   a) Polling cycle for the button matrix (this is used in all modes).
   b) Resets counter array.
   c) Displays buttons pressed on the LCD (this is used in normal mode).
   d) Pass buttons pressed via function-call return (this is used in the settings and entry modes).
   e) Keypad-driven interrupt handler logic (this is used in normal mode). Explained in a section below.
   f) Enable or disable the DAC driver.

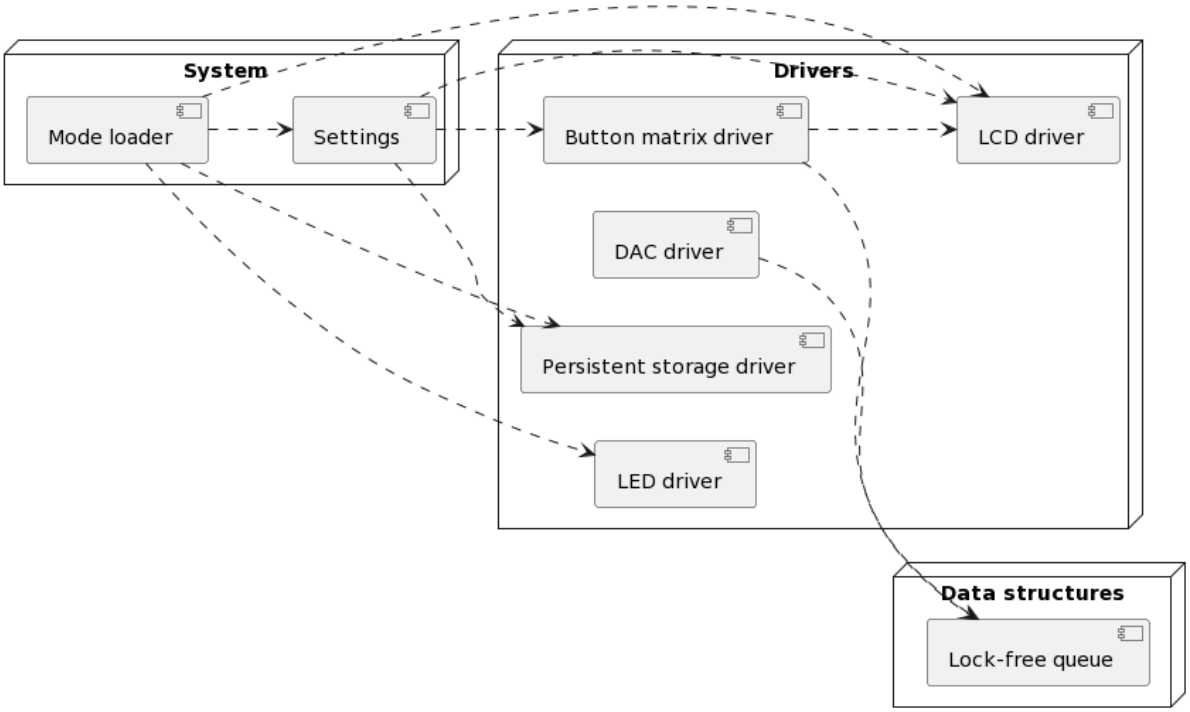

Figure 1: Software components proposed for the system, grouped by functionality. Dashed arrows between components indicate dependencies.

4. DAC driver Note that this component's code is meant to be encapsulated in a timer interrupt handler. This is the only way to reliably drive the DAC. The driver should have functionality allowing this handler to be disabled/enabled by the button matrix driver.
   a) Read settings related to tone generation, and adapt accordingly.
   b) Keep track of a global counter in between handler invocations in order to track tone generation progress.
   c) Keep tones being generated in a global array that is NOT torn down between invocations of the timer interrupt.
   d) Produce DTMF tone when the timer interrupt is invoked.
5. Persistent storage driver Note that this component is not as simple as it seems, since our microcontroller's flash memory will not have a filesystem.
   a) Serialize and persist data to flash memory.
   b) Load and de-serialize data from flash memory.
6. LED driver
   a) Turn on LED on command.
   b) Turn off LED on comamnd.
7. LCD driver
   a) Encodes and displays an arbitrary C string on the LCD.
   b) Encodes and displays DTMF symbols on the LCD.
   c) Deals with automatic scrolling of displayed text.

2.3 Interaction between the button matrix driver and the DAC driver

This interaction is particularly complex, and should be elucidated.

By default, the timer interrupt handler driving the DAC is disabled.

When a single invocation of the polling cycle detects one or more key presses, it performs the following operations:

1. It immediately displays the symbols pressed on the LCD.
2. It then pushes each symbol onto the lock-free queue in order of detection.
3. It checks if there is a running timer interrupt. If not, it starts one.
4. To start a timer interrupt, it determines the symbol's sampling frequency and loads the correct code. Resets global sample counter to 0.

The timer interrupt handler does the following:

1. Checks the global counter to see if it has to generate another tone sample.
2. If yes, generates the tone. Increment the global sample counter.
3. If no, checks the lock free queue. If this is not empty, it starts a new timer interrupt to generate the next tone.

How do we deal with multiple tones being generated at the same time?
- If two tones have same base frequency, they interfere with each other. - Even if not, they can still interfere as both are trying to drive the DAC, creating weird aliasing effects.
- If we use an output queue, how do we keep the system from getting out of sync with user's presses for long generation interval? - Also, how do we trigger the next timer interrupt in the queue? Piggybacking on the interrupt handler (if counter reached end, enable next interrupt).

2.4 Design phases

3 Management

Include a time plan. Show task dependencies. How are these going to be managed?

4 Closure

References

[1] B. Stroustrup, *The C++ Programming Language*. Addison-Wesley Professional, special ed., 2000.