Michael Daniel Werezak - 20303777

# Lab Assignment 2 - Finite State Machines

The design of the finite state machine created for this lab assignment can be divided into two major components (that shall be referred to in this report as modules).

The MorseReader module is responsible for interpreting input from the pushbutton as either dots or dashes. It handles rising and falling edge interrupts from EINT3 (External Interrupt Input 3 on the LPC1768 system) in order to receive pushbutton input, debounces that input, and then uses a measurement of the duration between when the push button was pressed to when it was released in order to determine if the input should be treated as a dot or a dash. In order to do this, the Timer and Pushbutton modules were created to act as a layer of abstraction between MorseReader and the timer/counters and GPIO functionality built into the LPC 1768.

The *FSM* module is responsible for creating an in-memory representation of the finite-state machine (FSM). This in-memory representation is necessary in order for the finite state machine to be properly generic and definable using only configuration or hardcoded values. Once this is done, calls from MorseReader indicating input are then received in order to operate the overall system.

## The MorseReader Module

The chief responsibilities of this module are input debouncing and recognition of dots and dashes. Both are accomplished by measuring the amount of time that has passed since the pushbutton is initially pressed using a single timer, TIM0. Since we only cared about the amount of time passed after a known event, TIM0 was configured to ignore the value in its Match Register completely. Since TIM0's frequency was left as 1/4 of the system core clock, the value of the Prescale Register was set so that TIM0's Timer Counter would read the number of milliseconds elapsed since the timer was last reset and started.

To debounce the input, MorseReader was programmed to reset the timer and ignore further interrupts from EINT3 for a debounce delay of 60 ms. This delay was experimentally determined by measuring the time between first press and last bounce using the code in debouncetest.c. Once the debounce delay has expired, MorseReader waits for next interrupt from EINT3 indicating that the button has been released and looks at TIM0 again. If the duration on the timer is greater than 500 ms, the event is treated as a dash, otherwise it is treated as a dot, and the FSM is updated before resetting the timer and returning to the ready state.

## The FSM Module

In order for the FSM to be configurable, an in-memory representation of the FSM must be created at initialization. To do this, a finite number of possible inputs were defined and assigned an integer index. For the morse code FSM, two inputs were used: FSM_DOT (assigned a value of 0), and FSM_DASH (assigned a value of 1). The in-memory representation of the finite state machine then consisted of a collection of linked nodes. Each node represents a state and is implemented as an FSMState data structure that contains an array of pointers indicating the next state to transition to depending on which input is applied (hence the need to assign each possible input an integer index).

Each FSMState node is stored in a large statically allocated array where the FSMState at index 0 is reserved for the initial state. The process of operating the finite state machine then consists of keeping

track of which state is the “current” one, looking up which FSMState to transition to when an input is applied, and then updating the pointer to the “current” FSMState. This process was implemented using an FSMRunner data structure, which is then used by MorseReader to operate the overall system.