Introduction

In the presentation, you should have learned the basic theory of state machines. In this activity, you will develop firmware implementing a simple state machine

Objective

In this lab, we will implement a 3-state state machine represented by the chart in Figure 1. The inputs to this state machine will be a pair of switches, used as digital inputs. The outputs of this state machine will be a set of 3 LEDs, with different patterns depending on the state.

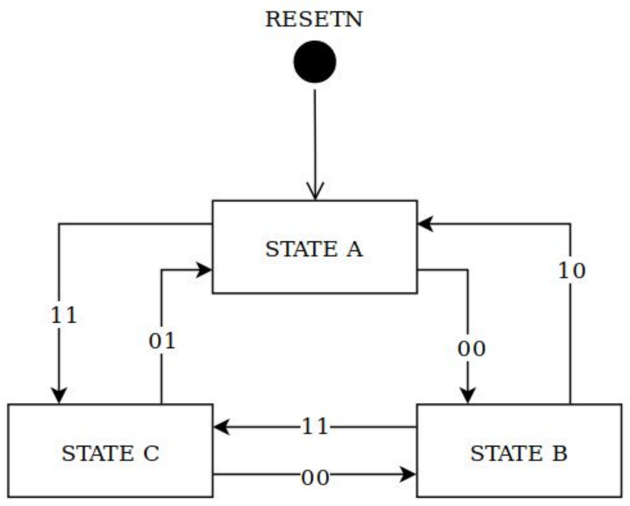

Figure 1. A state machine chart, with three states and 2 inputs

Background

State Machines

A state machine is an abstract representation of a computer system. The core concept involves a system, which can take on a variety of "states." Each state corresponds to a different output or function of the system. The machine can change between states based on the set of inputs given to it, and the knowledge of the current state.

Before writing code, it is often a good idea to draw a chart representing your state machine. Figure 1 depicts one such chart, with arrows representing transitions and boxes representing states. The black circle represents the default state, entered when the system is reset (pressing the reset button on your microcontroller). The arrows are labeled with binary digits corresponding to the combination of inputs required to trigger that transition. For example, the label of 10 means the first input must be 1, and the second input must be 0 to trigger the transition from State B to State A.

The first step in writing a state machine is to define your states. This can easily be done using a series of `#define` statements at the top of your program file. These are written in the form of `#define NAME <VALUE>`. This is good for names that have constant values, and functions like a find-and-replace function before your code actually gets downloaded onto the microcontroller.

We must also define a variable to hold the current state. This variable will normally be of an integer type (`int` or `byte`). The variable will then have its default value set. This defines where the state machine “starts” in its cycle.

We implement a state machine in Arduino using a `switch` statement in our `loop()` method. Each case in our `switch` statement represents one state. Thus, we put that state’s functionality in its corresponding `case`.

Finally, we must define our transitions from one state to another. This can be accomplished in the form of one or more `if` statements that check the state machine’s inputs, make appropriate calculations and decisions, and set the state variable accordingly. Upon the next entry into the `switch` statement, the state variable will cause the program to enter the desired state.

Instructions

1. Wire up your microcontroller with two switches as digital inputs, and three LEDs as digital outputs.
2. We will be writing an implementation of the state machine specified in Figure 1. Begin by defining the specified states in the figure.
3. Build the functionalities for each state by writing a function for each one. Each state's function will be lighting up an LED corresponding to that state, and turning off all others. For example, being in STATE A will light up the first LED and turn off the second and third.
4. Write a `switch` statement to represent the overall state machine's structure. This should have 3 cases, representing our 3 states. Remember to call each function and include a `break` statement.
5. In each `case`, use `if` statements to define the transitions to the next state in accordance with the chart in Figure 1.
6. Test your state machine and debug as necessary
7. **EXTENDED ACTIVITY:** If your initial state machine works, try the following:
    a. Cause the LEDs to flash, with a different blink rate based on the state.
        i. You will have to use some kind of `delay` function for this.
    b. Use interrupts to read the switch positions and change states
        i. Ensure that your switches are wired to interrupt pins (these have ~ next to them)
        ii. Develop a function that reads the switch combinations and sets the global state variable to the desired value
        iii. Attach this function to pin change interrupts on both switches

Code Functions

Main Activity

- `digitalWrite(pin, value)`
  - Writes a high or low voltage to a specified pin
  - Arduino Website Documentation
- `digitalRead(pin)`
  - Returns `true` if the selected pin is `high`, or false otherwise.
    - Used to read the position of the switch
  - Arduino Website Documentation

Extended Activity

- `delay(milliseconds)`
  - Stops program execution for a specified number of milliseconds
  - Arduino Website Documentation
- `attachInterrupt(interrupt, ISR, trigger)`
  - Causes the function `ISR` to be called upon a `trigger` event on specified `interrupt`
  - Arduino Website Documentation
- `digitalPinToInterrupt(pin)`
  - Converts a specified pin number into its corresponding interrupt number to be used in `attachInterrupt`
  - Arduino Website Documentation