Button Design

1 Summary

The keypad is required to provide user inputs to the calculator. In accordance with the requirements of the calculator the intended operation of the keypad is as follows:

1. Enter up to 4 digits, no more digits than this will be registered (these will be displayed upon the LED as entered)
2. Press either the square root or integer power function buttons. Once pressed the calculator will automatically perform the selected function.
3. Square root function will automatically execute and display the result.
4. Integer power function will wait until one more digit is pressed (this selects the exponent) and then automatically execute the function and display the result.
5. Press either function button again to reset the calculator.

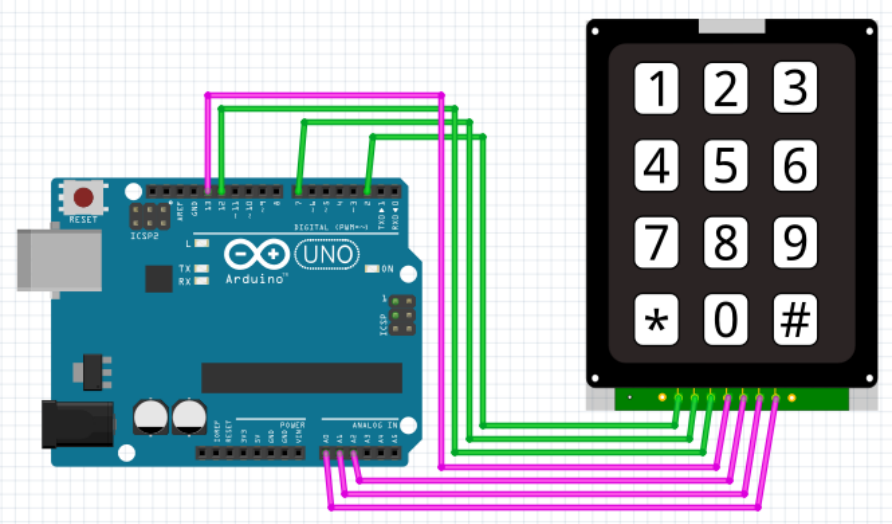

Figure 1: Calculator keypad wiring diagram.

Figure 1 illustrates the proposed keypad design. A prefabricated 3x4 keypad will be used as this simplifies the calculator construction and has all the required buttons. The keypad is operated using digital I/O pins on the Arduino of which there are 4 remaining after integrating the LED. This means that the remaining 3 connections must be made by using analog pins as digital pins via the Arduino's internal ADC. The Arduino interprets button presses by directly reading the column (green wires in Figure 1) of the pressed button and then scanning through the rows (pink wires in Figure 1) to determine its position in the matrix.

To ensure that the keypad is robust unideal inputs and pressing buttons simultaneously, the Arduino will only register button releases and has a wait time of 0.1 seconds before another button press can be registered. The keypad will also utilize a software debouncing algorithm which repeatedly checks the state of the button and resets a 20ms timer every time the state changes.

2 Bill of Materials

Components	Supplier	Supplier Part Number	QTY	Unit Price
Keypad	Digikey	1528-1136-ND	1	5.34
Arduino Uno	N/A	N/A	1	Provided
Connections		Negligible for prototype.		
			GRAND TOTAL	5.34

3 Requirements and Specifications

- Controlled by an Arduino Uno.
- 3x4 keypad. 12 buttons - digits 0-9 and selection buttons for integer power and square root functions.
- Must not exceed $5.886 AUD (This is the remainder of the original $10 AUD budget after designing display interface).

4 Arduino Uno

Figure 2 shows the pinout for the Arduino Uno. The Arduino Uno has 14 digital I/O pins. 6 of these have inbuilt hardware to produce PWM signals. These pins are 3, 5, 6, 9, 10, and 11. There are also a further 6 analog I/O pins. These can be operated as digital pins; however, this comes at a cost of increased power consumption through the ADC.

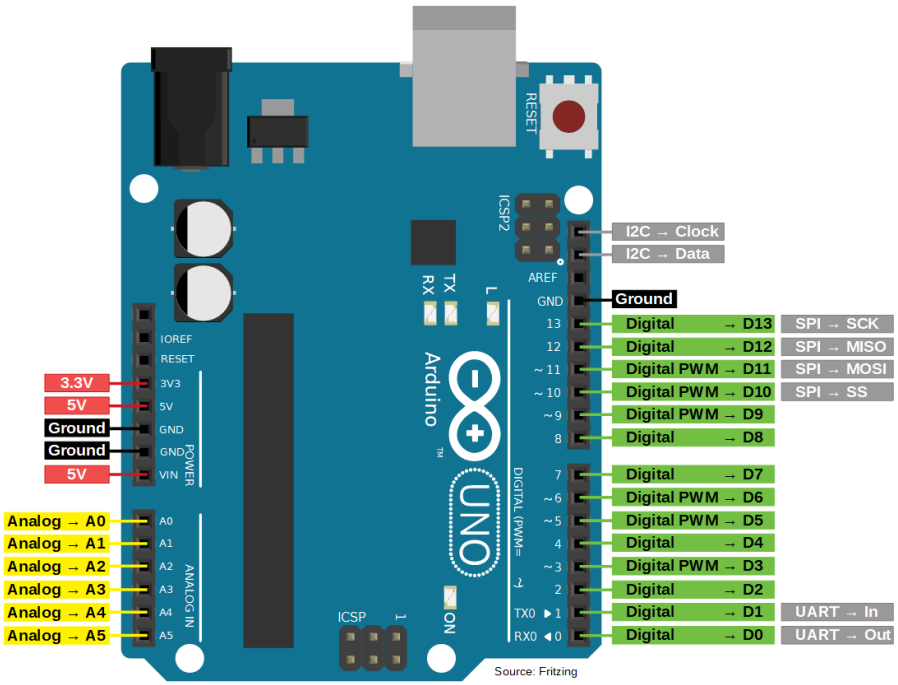

Figure 2: Arduino Uno pin map.

Table 1: Arduino Uno major specifications [1].

Microcontroller	**ATmega328**
Input Voltage	7V – 12V

Number of Digital I/O Pins	14 (6 of which can provide hardware PWM signal)
Number of Analog I/O Pins	6
Maximum Current Per I/O Pin	40mA
Maximum Current on 3.3V Supply Pin	50mA
Maximum Current on 5V Supply Pin	500mA
Maximum Combined I/O Current Draw	200mA
Maximum Current Available to Arduino Through USB Supply	500mA
Maximum Current Available to Arduino Through Barrel Power Connector Supply	1A
Flash Memory	32 KB
SRAM	2 KB
EEPROM	1 KB
Clock Speed	16 MHz
I/O Internal Pull-Up Resistors (Does not have internal pull-down resistors)	20 $K\Omega$ – 50 $K\Omega$ (automatically disconnected and disconnect on reset/boot).
I/O Bootup State	Set to input.
Boot Time (progress from boot loader to installed code on powerup)	6 – 8 seconds.
ADC Current Consumption	9.2mA

5 Keypad Hardware Selection

The Adafruit membrane 3x4 matrix keypad will be used.

Figure 3: Adafruit keypad.

This keypad has the 12 required buttons with labelling and is prefabricated with female wire connectors preinstalled for easy integration with the Arduino Uno. It is also the cheapest available keypad and the only one which fits within the given budget.

Constructing a custom keypad was considered but is highly unfeasible. The cheapest possible custom keypad that can be constructed using components sourced from Digikey is detailed in Table 2, which exceeds the budget. This will also complicate the construction of the calculator and introduce more room for error. For these reasons, the prefabricated Adafruit keypad has been selected.

Table 2: Bill of materials to construct a custom keypad.

Components	Supplier	Supplier Part Number	QTY	Unit Price
Button	Digikey	563-1932-5-ND	12	0.654
Perf Board	Digikey	V2018-ND	1	8.19
Connections		Negligible for prototype		
			GRAND TOTAL	16.038

6 Keypad Construction and Software Integration

6.1 Internal Construction

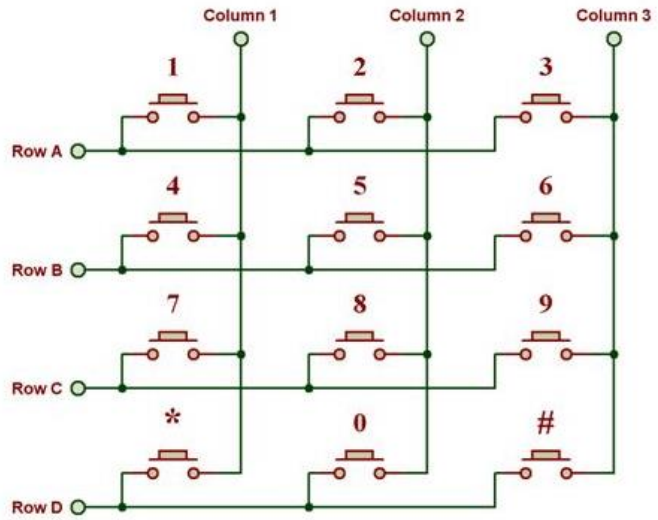

Figure 4: Internal construction of keypad [12].

The internal construction of a keypad is illustrated in Figure 4. The buttons on a keypad are arranged in a matrix with 3 columns and 4 rows. There is a wire bus for each row and each column, giving 7 in total. Each button is a normally open single pole single throw (SPDT NO) button. When a button is pressed it will complete a circuit between the button's respective row and column, allowing the pushed button to be located and specified.

6.2 Arduino and Keypad Communication

The Arduino and keypad will interact through the Arduino's I/O pins. This will require 1 pin for each bus, meaning 7 in total for the 3x4 keypad. The pressed button will be determined using a scanning algorithm that runs through the columns and rows of the button matrix.

In the default state the Arduino will set all the row buses to the LOW pin state while the column buses will be fixed to a HIGH pin state using the Arduino's internal pullup resistors. When a button is pressed the column pin for that button is pulled LOW by the grounded row bus, therefore the Arduino can register a LOW input reading on the respective column pin. Using the known column, the Arduino will proceed to scan for button's the corresponding row.

The pin states of the Arduino are floating during the powerup and boot stages. This means that the Arduino has the potential to register button presses during this phase. Ideally, hardware pull-up and pull-down resistors would be utilized to tie the states of the pins as required, however, there is no room in the budget to include these resistors. As no current can flow due to the buttons being open there will be no unwanted power consumption. Also, if the keypad is initialized in the software before the LED display there is no chance of displaying unwanted numbers. For these reasons this floating stage will have negligible impact upon the performance of the calculator.

All the row pins will be set to LOW outputs and the column pins will be set to HIGH inputs. The algorithm will then cycle through each row pin, setting it to HIGH and observing the input at the verified column pin. If the column pin reads LOW the pressed button is not in the row, if it reads HIGH then the pressed button is in the row and the pressed button is identified. This algorithm is shown in Figure 5 [2].

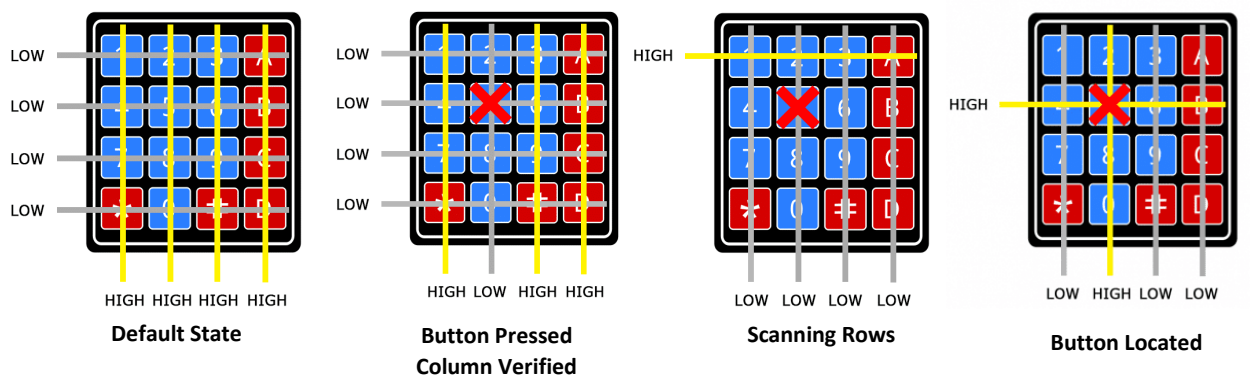

Figure 5: Button detection algorithm.

6.3 Software

50ms is the average time which a person holds down a key on a keyboard while typing [3]. This means that the button locating algorithm described in section 6.2 and debouncing algorithm (section 8.3) must not exceed 50ms. Assuming a worst-case scenario bounce time of 6.2ms [4] with a debounce delay of 10ms the button locating algorithm will have up to 33.8ms to scan for the correct button.

The longest operation is detecting a button in row 4. This involves five operations, reading the column and then scanning through the 4 rows. Each row pin will also have to be set to high once to locate the row. The inbuilt Arduino functions digitalWrite and digitalRead take 3.40ms and 4.9μs respectively [5] [6]. Therefore, if the two digitalWrite and digitalRead functions are used in our worst-case scenario where the pressed button is in row 4, these functions will only take up 15.6ms. Given that our algorithm has 33.8ms to operate, that leaves a contingency of up to 18.2ms. The Arduino has a 16 MHz [1] clock speed meaning that 291,200 clock cycles will occur in 18.2ms. This is sufficient time for the Arduino to execute the required operations.

7 Power Consumption

The maximum power draw of the keypad circuit is dependent upon the duration of time the button is pressed for. The internal resistance of the button's contacts will be assumed to be the same as that listed in the data sheet for a similar membrane keypad. This value is 200 Ohm [7]. In accordance with Ohms law this means that when a button is pressed 25mA will be drawn, giving a maximum power consumption of 125mW.

The use of digital I/O pins has been prioritized for improved power efficiency. There are 3 excess pins on the on the keypad to the number of available digital I/O pins on the Arduino, hence these must be connected to the Arduino's analog I/O pins. This requires the use of the Arduino's onboard ADC. The Arduino uses a 10-bit SAR ADC. These commonly have a power consumption of 7mW [8].

This means the maximum power consumption of the keypad is 132mW.

8 Button Bounce

Due to the physical nature of a mechanical button, inputs from buttons or switches often oscillate for a brief period before settling to the correct state. These oscillations can be registered by the Arduino and

cause unwanted operations to occur in the program. Results tested by Jack Ganssle revealed that an average bounce time for an assortment of 18 switches was 1.6ms with a maximum of 6.2ms [4].

The Arduino will be registering the button presses as inputs, therefore if the time it takes to read an input is not far greater than the bounce time of a button the Arduino will likely get faulty readings. The digitalRead function takes approximately 5ms [9]. This is less than the possible 6.2ms bounce time meaning some form of hardware or software debouncing will be required.

8.1 Hardware Debounce

There are many options for hardware debouncing with the most common being the addition of a series capacitor and resistor in parallel with the switch. This RC circuit will smooth out the transition between the high and low voltages by discharging or charging the capacitor over the button bounce time, depending on if the button has been made or opened. This transition time is determined by the RC time constant of the circuit τ.

This hardware debounce solution can be improved by the addition of a Schmitt trigger. As the capacitor is charging or discharging the input pin will experience voltages in between 0V and 5V. The Arduino could then attempt to read this *in-between voltage input* and give nonsense readings. The addition of a Schmitt trigger removes the ambiguity of this charging or discharging time. A Schmitt trigger is fundamental an Op Amp with positive feedback. It has two thresh hold voltages, input voltage high threshold and input voltage low threshold. When the input voltage rises from below the low threshold to the high threshold, the Schmitt trigger will flip its output and when the input then drops from above the high thresh hold to the low threshold the output will again flip. This effectively creates a *dead zone* in between the high and low voltages where voltages can oscillate without effecting the output.

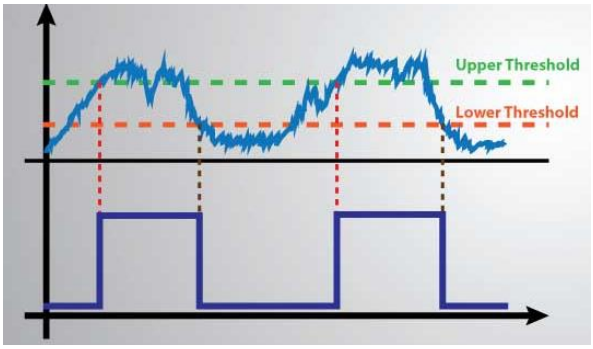

Figure 6: Conversion of analog signal to digital signal using Schmitt trigger debouncing circuit [10].

Hardware debounce can be the best option for projects that are not controlled by a micro controller and do not have many buttons. However, the addition of capacitors and resistors will increase the cost of components and will require more physical space. For this project, a hardware debouncing solution is not feasible as the budget does not have the capacity to include the required components.

8.2 Software debounce

Software debounce solutions can be implemented in situations where the input is being registered by an intelligent device such as a micro controller or single board computer. It offers a cost-effective of debouncing button signals as no extra components are required.

In its simplest form, software can debounce a switch by first registering that a button has been pressed and then waiting a set time before reading the value. This set time must be longer than the bouncing period so that the reading is not taken while the bouncing is occurring. This process is shown in Figure 7.

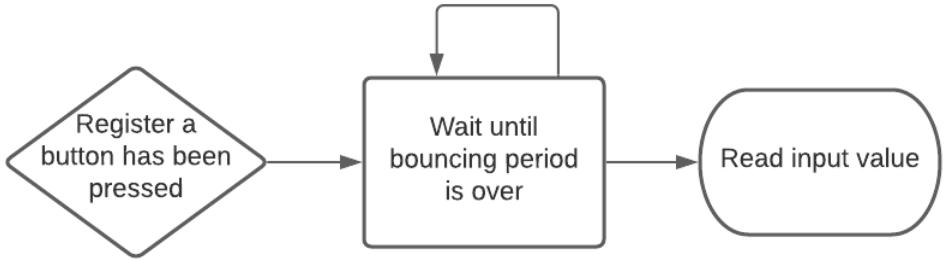

Figure 7: Software debouncing algorithm. Register a change of state and wait a set period before reading the set state.

This solution while simple and easy to implement has its downfalls. One, a long delay to read the input could make the system feel sluggish. Two, the user could have let go of the button before the input is read. For the calculator design an improved software debounce will be used.

8.3 Debounce solution

The debounce solution for the keypad will utilize code adapted from the built in Arduino debounce code made by David A. Mellis, Limor Fried and Arturo Guadalupi [11]. This will reduce development time and is a proven reliable and fast debouncing method. The Arduino constantly compares the current state of the input pin to the previous state. If there is a change in state this means the button has been pressed or is bouncing.

When this is detected a bounce timer will begin to count from zero. The bounce timer records the time that has passed since the input has changed. The code will then check if this bounce timer surpasses a value called bounce delay. Bounce delay is a set value which determines the time that the Arduino will wait to validate the value of the pin, therefore the bounce delay must not be smaller than the time in between bounces.

In a study on keypress timings by Jeffrey D. Allen it was found that on average keypress duration for people entering text on a keyboard was 50ms-300ms [3]. Therefore, only inputs longer than 50ms will be registered. In another study Jack Ganselle tested 18 switches and recorded button bounce time. Ignoring two outliers which had significant bounce times it was found that no switch exceeded 6.2ms [4]. There is no given bounce time on the selected keypad's bounce duration. It can be assumed that it will be similar to the time given on the data sheet for a similar membrane keypad which is listed as having a bounce time of 5ms [7]. This value fits well with Jack Gansalle's findings so the design will consider 10ms to be the maximum bounce time for the keypad switches. Therefore, the longest debounce time expected is 20ms (bounce delay + max bounce time).

There are tradeoffs when selecting the bounce delay value. Too small of a value may not completely debounce the input. Too large of a value may lead to the button being released for the reading is completed. The bounce delay value selected will be 10ms. 10ms is a conservative value close to double the maximum bounce time recorded by Jack Ganselle, meaning there should be no sampling during the bounce time [4]. A bounce delay of 10ms will also give the keypad scanning algorithm surplus time to read

the input, up to 33.8 assuming a worst-case scenario bounce time of 6.2ms. This allows the design to meet the requirement of reading presses that have a duration of 50ms or above. The algorithm is illustrated below in Figure 8.

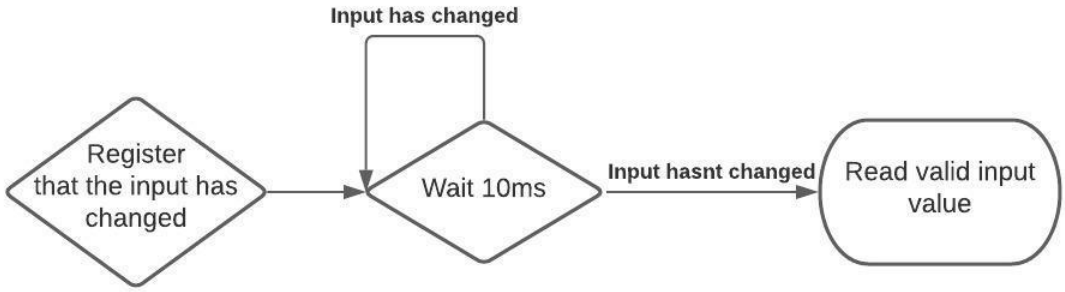

Figure 8: Debounce algorithm.

9 Bibliography

[1] Atmel, "ATmega328P DATASHEET," Atmel, 2015. [Online]. Available: https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf. [Accessed 29 March 2021].

[2] K. Pattabiraman, "HOW TO SET UP A KEYPAD ON AN ARDUINO," 2017. [Online]. Available: https://www.circuitbasics.com/how-to-set-up-a-keypad-on-an-arduino/. [Accessed 16 April 2021].

[3] K. K. a. R. Maxion, "Keystroke Dynamics - Benchmark Data Set," 2 July 2009 . [Online]. Available: http://www.cs.cmu.edu/~keystroke/#ref1. [Accessed 17 April 2021].

[4] J. Ganselle, "A Guide to Debouncing, or, How to Debounce a Contact in Two Easy Pages, by Jack Ganssle," 2014. [Online]. Available: http://www.ganssle.com/debouncing.htm. [Accessed 14 April 2021].

[5] Robotics Back End, "Arduino Fast digitalWrite," [Online]. Available: https://roboticsbackend.com/arduino-fast-digitalwrite/#:~:text=We%20have%20the%20answer%3A%20a,on%20an%20Arduino%20Uno%20board.. [Accessed 17 April 2021].

[6] dannyelectronics, "The price of AVR Arduino," 1 May 2016. [Online]. Available: https://dannyelectronics.wordpress.com/2016/05/01/the-price-of-avr-arduino/. [Accessed 17 April 2021].

[7] "4x4 Matrix Membrane Keypad," [Online]. Available: http://www.electronicoscaldas.com/datasheet/Teclado-membrana-matricial-4x4.pdf. [Accessed 17 April 2021].

[8] A. Walsh, "Explaining SAR ADC Power Specifications," Analog Devices, [Online]. Available: https://www.analog.com/en/technical-articles/explaining-sar-adc-power-specifications.html. [Accessed 18 April 2021].

[9] SukkoPerra, "How exactly slow is digitalRead()?," 15 July 2015. [Online]. Available: https://forum.arduino.cc/t/how-exactly-slow-is-digitalread/325458. [Accessed 14 April 2021].

[10] Dejan, "What is Schmitt Trigger | How It Works," [Online]. Available: https://howtomechatronics.com/how-it-works/electrical-engineering/schmitt-trigger/. [Accessed 14 April 2021].

[11] ARDUINO, "Debounce," 30 August 2016. [Online]. Available: https://www.arduino.cc/en/Tutorial/BuiltInExamples/Debounce. [Accessed 17 April 2021].

[12] Engineers Garage, "How to interface 4×3 (Alphanumeric) keypad with Arduino uno : Arduino Alphanumeric keypad programming," Engineers Garage, 26 July 2019. [Online]. Available:

https://www.engineersgarage.com/arduino/alphanumeric-keypad-with-arduino/. [Accessed 16 April 2021].