Indian Institute of Science Education and Research (IISER), Pune
PH3144 Electronics-I with Lab
Nikhil Sharma 20246701
Quantum Gates using Arduino Uno
Project Report Group #5

Introduction

Our project deals with the functioning of Quantum Gates. We have built a small system that essentially encompasses three qubits as input, allowing us to apply a user-selected gate, which then gives a processed output with its corresponding phase. This provides a very hands-on experience of how certain Quantum Gates function when classical inputs are applied, showing us a visible shift in either the bit and/or the phase. This project has been incredibly engaging and motivating, as we develop a one-of-a-kind tool—or a "toy," in loose terms—that lets us experiment with inputs and learn about various Quantum Gates in a fun and interactive manner.

In building this project, we combined multiple skills to create an ensemble of electronics powered by a C program, alongside basic knowledge of Quantum Gates and Qubits. This experience allowed us to integrate both practical electronics and theoretical quantum concepts into a cohesive system. The theoretical background to these topics, along with the detailed circuit schematic, is provided in the following sections of the report. Additionally, we have outlined methods to further enhance the system, aiming to make it even more enjoyable and interactive for students and researchers alike.

Theoretical Background

1. **X Gate (Pauli-X or NOT Gate)**

- **Matrix**:

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

- **Function**: The X gate flips the state of a qubit, converting $|0\rangle$ to $|1\rangle$ and vice versa.
- **Effect**: It's equivalent to a classical NOT gate.
- **Action**: $X|0\rangle = |1\rangle$, $X|1\rangle = |0\rangle$

2. **Y Gate (Pauli-Y Gate)**

- **Matrix**:

$$Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}$$

- **Function**: The Y gate flips the qubit state like the X gate but also introduces a phase shift.
- **Effect**: It changes $|0\rangle$ to $i|1\rangle$ and $|1\rangle$ to $-i|0\rangle$.
- **Action**: $Y|0\rangle = i|1\rangle$, $Y|1\rangle = -i|0\rangle$

3. **Z Gate (Pauli-Z Gate)**

- **Matrix**:

$$Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

- **Function**: The Z gate introduces a 180° (π radians) phase shift to the $|1\rangle$ state, leaving $|0\rangle$ unchanged.
- **Effect**: It's often called a phase flip.
- **Action**: $Z|0\rangle = |0\rangle$, $Z|1\rangle = -|1\rangle$

4. **S Gate (Phase Gate)**

- **Matrix**:

$$S = \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix}$$

- **Function**: The S gate applies a 90° (π/2 radians) phase shift to the $|1\rangle$ state.
- **Effect**: Used to incrementally shift the phase of a qubit.
- **Action**: $S|0\rangle = |0\rangle$, $S|1\rangle = i|1\rangle$

5. **T Gate (π/4 Phase Gate)**

- **Matrix**:

$$T = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & \frac{1}{\sqrt{2}} + i\frac{1}{\sqrt{2}} \end{pmatrix}$$

- **Function**: The T gate applies a 45° (π/4 radians) phase shift to the $|1\rangle$ state.
- **Effect**: Often used for creating finer phase adjustments.
- **Action**: $T|0\rangle = |0\rangle$, $T|1\rangle = e^{i\pi/4}|1\rangle$

6. **H Gate (Hadamard Gate)**

- **Matrix**:

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

- **Function**: The Hadamard gate creates superposition by converting $|0\rangle$ to $\frac{|0\rangle+|1\rangle}{\sqrt{2}}$ and $|1\rangle$ to $\frac{|0\rangle-|1\rangle}{\sqrt{2}}$.
- **Effect**: It's used to prepare qubits in superposition states, where each basis state has an equal probability.
- **Action**:
  - $H|0\rangle = \frac{|0\rangle+|1\rangle}{\sqrt{2}}$
  - $H|1\rangle = \frac{|0\rangle-|1\rangle}{\sqrt{2}}$

Detailed Description

There are 5 most important components to this project, namely Arduino Uno, LED bulbs, SPDT switches, LCD 16x2 Display, and a Rotary Encoder. All these were joined by connector wires, most of which were male-to-male type.

To start with the buildup, we first started with brainstorming about a physical model and the various components we will need, to build a robust and efficient system to get our desired output.

Initially, we started off with an Arduino UNO, as we needed a more robust Arduino, instead of nano. Now, the user-desired input needed to be provided to the system, along with a random phase, which is also easily visible to the observer, so we decided to plug in 3 LEDs to mimic a 3-qubit system, which can be turned on or off using a DPDT type switch. The turning on or off of the switch does two things, one is turning on the LEDs so the input is bright and clear, as well as registering an input for the corresponding qubit state, 0 or 1. The final switch on-off pattern corresponds to a 3-qubit state input, for example, 001, 101, 111, etc.

Now, the next step is to add a random phase to the 3-qubit system and be able to choose a particular gate, out of H, S, T, X, Y, Z. We used a Rotary encoder for both these tasks. The push button on the encoder takes a random phase, anything between 0 to pi/2 which is about 1.57. The rotation of the pin switches between various gates that we have mentioned already, S, T, H, X, Y, and Z.

Following this, we need to display all these things, including the input qubits, the gate chosen, the random phase, and the corresponding outputs. We use a 16x2 LCD for this purpose, which is large enough for us to display all the necessary inputs and outputs in a decluttered fashion.

All these components are connected to an Arduino Uno board, on different digital pins, thus working in harmony to perform the gate on the corresponding qubit system and give us the output.

The working of the project is as follows for each corresponding gate with certain complimentary examples: (Assumption: The first qubit is the control qubit in all the controlled gates out of the many)

***X Gate**:* We input a random 3-qubit state, |101> let's say, and choose the X gate on the encoder. Now we push the encoder button to input a random phase, which could be anything between 0 to pi/2. The Arduino applies the bit flip for each qubit, as given in the theoretical description. We get the corresponding output, |010> with no change in the phase difference.

***Y Gate**:* We input a random 3-qubit state, |101> let's say, and choose the Y gate on the encoder. Now we push the encoder button to input a random phase, which could be anything between 0 to pi/2. The Arduino applies the bit flip for each qubit, as given in the theoretical description. We get the corresponding output, |010> with a complete change in the phase, adding a negative sign in the input phase since the controlled qubit is in state 1. There would be no flip in the phase if the controlled qubit was in state 0.

Z Gate: We input a random 3-qubit state, |101> let's say, and choose the Z gate on the encoder. Now we push the encoder button to input a random phase, which could be anything between 0 to pi/2. The Arduino does not apply the bit flip for each qubit, as given in the theoretical description. We get the corresponding output, |101> with no change in the phase.

***S Gate**:* We input a random 3-qubit state, |101> let's say, and choose the S gate on the encoder. Now we push the encoder button to input a random phase, which could be anything between 0 to pi/2. Now, the S gate does not flip the qubit states. Since the controlled qubit is 1 in this case, there is a multiplication of pi/2 phase factor in the final state, with all the qubit outputs remaining the same, thus making it |101> with phase

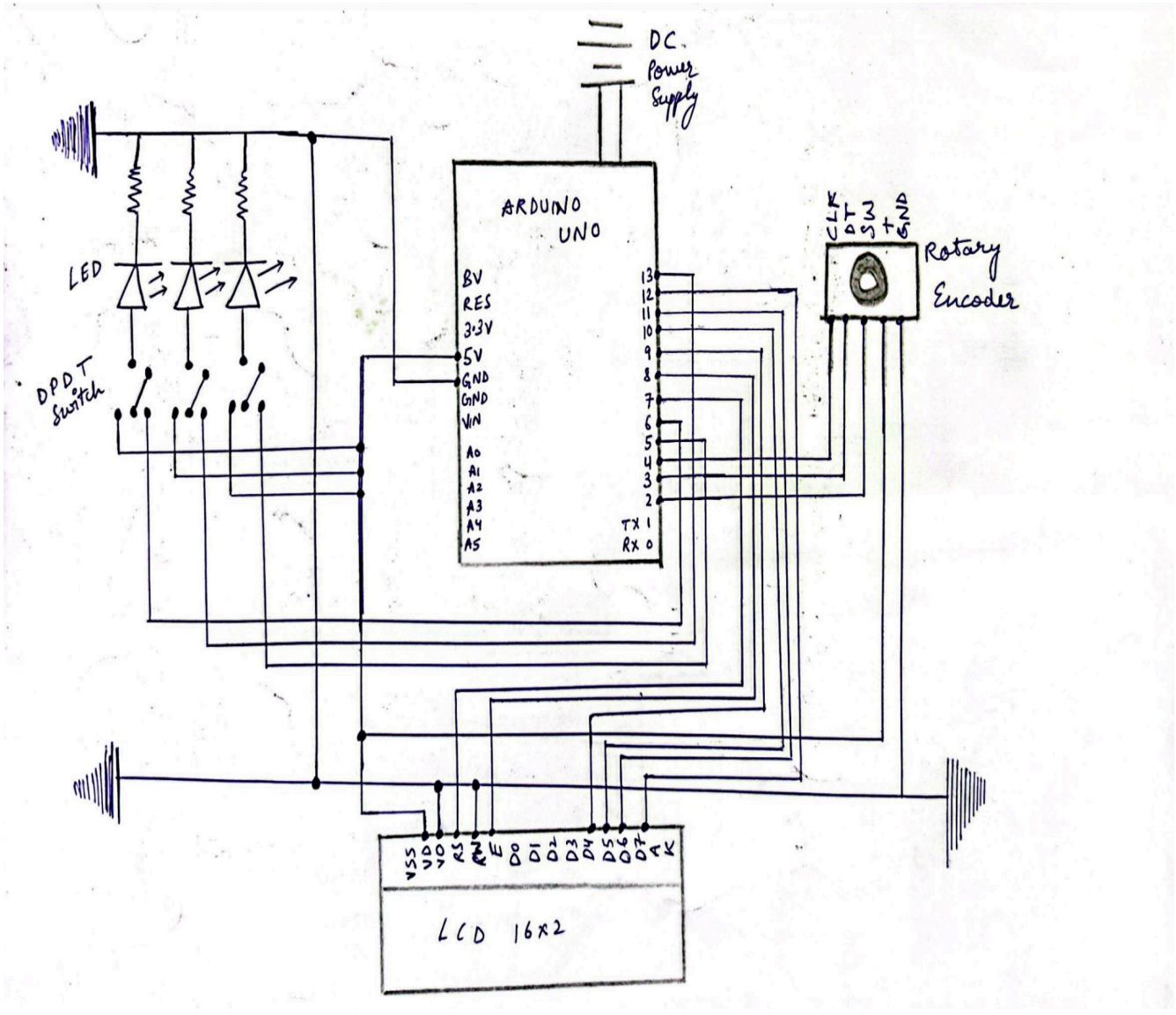
e^i(randomPhase+pi/2). There would be no phase difference if the controlled qubit was 0.

T Gate: We input a random 3-qubit state, |101> let's say, and choose the T gate on the encoder. Now we push the encoder button to input a random phase, which could be anything between 0 to pi/2. Now, the T gate does not flip the qubit states. Since the controlled qubit is 1 in this case, there is a multiplication of pi/4 phase factor in the final state, with all the qubit outputs remaining the same, thus making it |101> with phase e^i(randomPhase+pi/4). There would be no phase difference if the controlled qubit was 0.

Nikhil Sharma

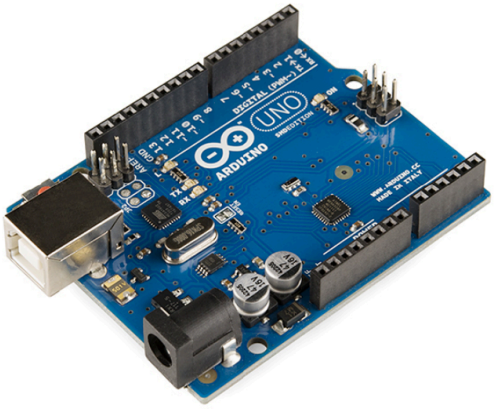
Circuit Diagram and Connections

Circuit Diagram

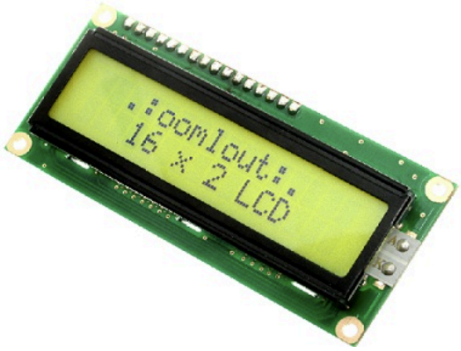


Nikhil Sharma

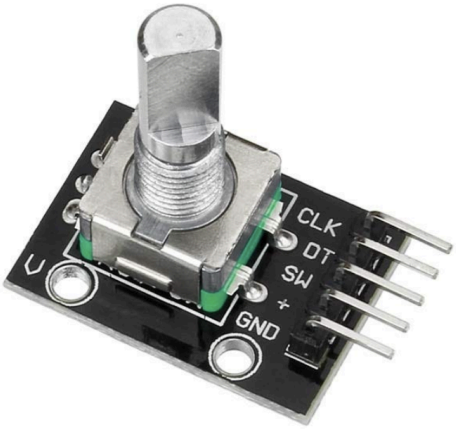
Component pictures



Arduino Uno for the main control of the system



LCD 16x2 for the display of input and output

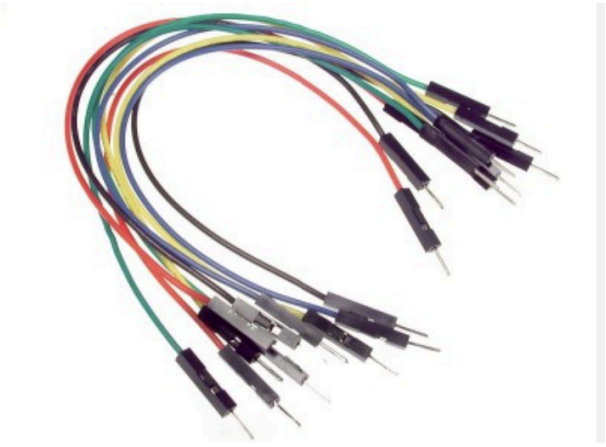


Rotary Encoder for switching between gates and choosing a random phase

..

LEDs for visibly highlighting input

DPDT Switch for giving input an lighting the LED



Wires to facilitate all the connections

Nikhil Sharma

Connection Table

Device: Rotary Encoder	Corresponding Arduino Pin
CLK	D4
DT	D3
SW	D2
+	5V
GND	Ground
Device: LCD 16x2 Display	
VDD	5V
VO	Ground
RS	D7
RW	Ground
E	D8
D4	D9
D5	D10
D6	D11
D7	D12

Device: LEDs	
Anode	5V
Cathode	Ground
Device: DPDT Switch	
Pin 1	5V
Pin 2	LED
Pin 3	Digital Pin
Device: BreadBoard	
2nd last line both sides	5V
Last line both sides	GND

Code with structural comments

```
#include <LiquidCrystal.h>
#include <RotaryEncoder.h>
//#include <stdio.h>
//#include <string.h>

// Define switch pins for classical input (LEDs)
const int switchPin1 = 6;
const int switchPin2 = 13;
const int switchPin3 = 5;

// Define LED pins
const int ledPin1 = 6;
const int ledPin2 = 13;
const int ledPin3 = 5;

// Rotary Encoder pins
const int encoderPinA = 4;
const int encoderPinB = 3;
const int buttonPin = 2;

// Initialize RotaryEncoder object
RotaryEncoder encoder(encoderPinA, encoderPinB);
```

```
// Define LCD pins: RS, E, D4, D5, D6, D7
LiquidCrystal lcd(7, 8, 9, 10, 11, 12);  // Modify these pins as per your wiring

// Variables for storing classical inputs
int qubit1, qubit2, qubit3;
int selectedGate = 0;  // Gate selection (0 to 5 for X, Y, Z, H, S, T)
float randomPhase;     // Random phase between 0 and pi/2

// Gate names
const char* gates[] = {"X", "Y", "Z", "H", "S", "T"};

// Function to generate a random phase between 0 and pi/2
float generateRandomPhase() {
  return random(0, 157) / 100.0; // π/2 ≈ 1.57, scale by 100
}

// Setup function
void setup() {
  // Initialize switch and LED pins
  pinMode(switchPin1, INPUT);
  pinMode(switchPin2, INPUT);
  pinMode(switchPin3, INPUT);

  pinMode(ledPin1, OUTPUT);
  pinMode(ledPin2, OUTPUT);
```

```
  pinMode(ledPin3, OUTPUT);

  // Initialize LCD
  lcd.begin(16, 2);  // Initialize 16x2 LCD

  // Initialize serial monitor for debugging
  Serial.begin(9600);

  // Seed random generator with a unique value
  randomSeed(analogRead(A0));

  // Initialize rotary encoder
  pinMode(buttonPin, INPUT_PULLUP);  // Rotary encoder push button
}

// Function to display each section sequentially on the LCD
void displayLCDSection(int q1, int q2, int q3, const char* gate, float phase, int result) {

  lcd.clear();

  float total_phase=0.00;

  if (q1 == 1 && gate == "S"){
    total_phase = phase+1.57;
  }
```

```
else if (q1 == 1 && gate == "T"){
  total_phase = phase+0.79;
}
else {
  total_phase = phase;
}

// Step 1: Display qubits and gate selection
lcd.setCursor(0, 0);
lcd.print("Q:");
lcd.print(q1);
lcd.print(q2);
lcd.print(q3);



lcd.setCursor(6, 0);
lcd.print("G:");
lcd.print(gate);
  // 3-second delay

// Step 2: Display phase
//lcd.clear();
lcd.setCursor(10, 0);
lcd.print("P:");
lcd.print(phase, 2);  // Show phase with two decimal places
```

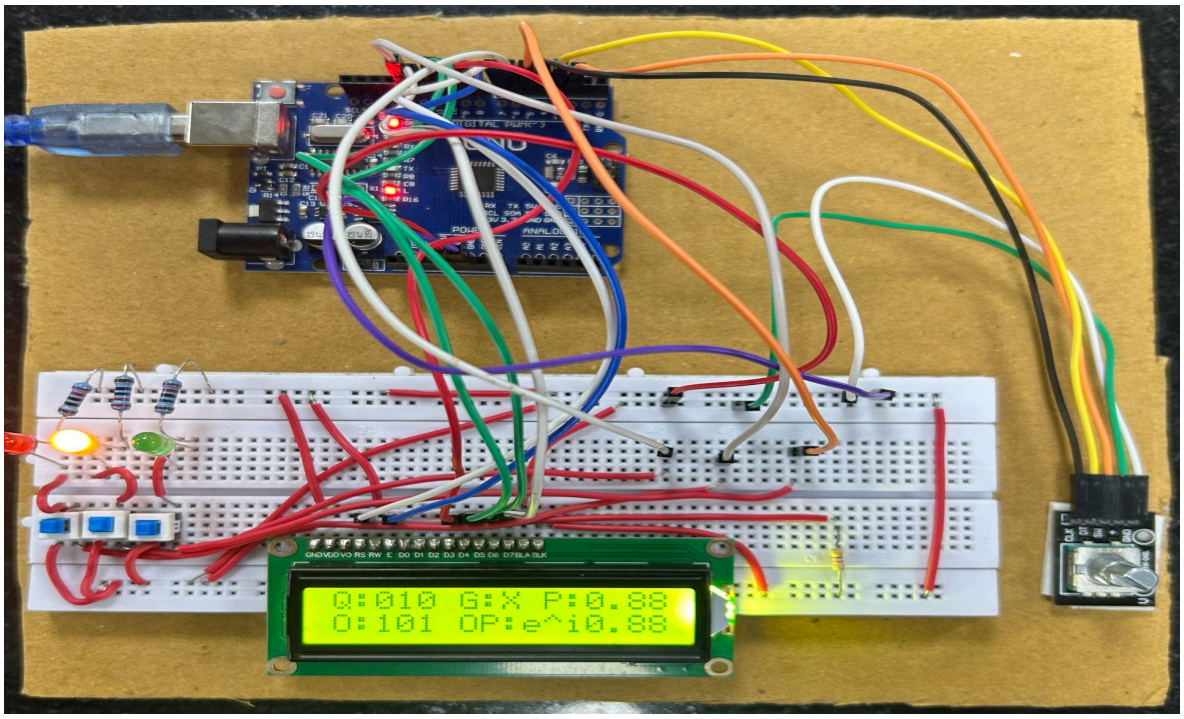
```
    // 3-second delay

  // Step 3: Display output
  //lcd.clear();
  lcd.setCursor(0, 1);
  lcd.print("O:");
  lcd.print(result, BIN);  // Output as a binary number (0 to 7)
    // 3-second delay

  lcd.setCursor(6, 1);
  lcd.print("OP:e^i");
  lcd.print(total_phase, 2);

}

// Quantum gate operations for 3-bit input
int applyGate(int gate, int q1, int q2, int q3, float phase) {
  //float added_phase = 0.00;
  int result = (q1 << 2) | (q2 << 1) | q3; // 3-bit input represented as integer (0-7)

  switch(gate) {
    case 0: // X Gate (NOT)
      result ^= 0b111;  // Flip all bits
      break;
```

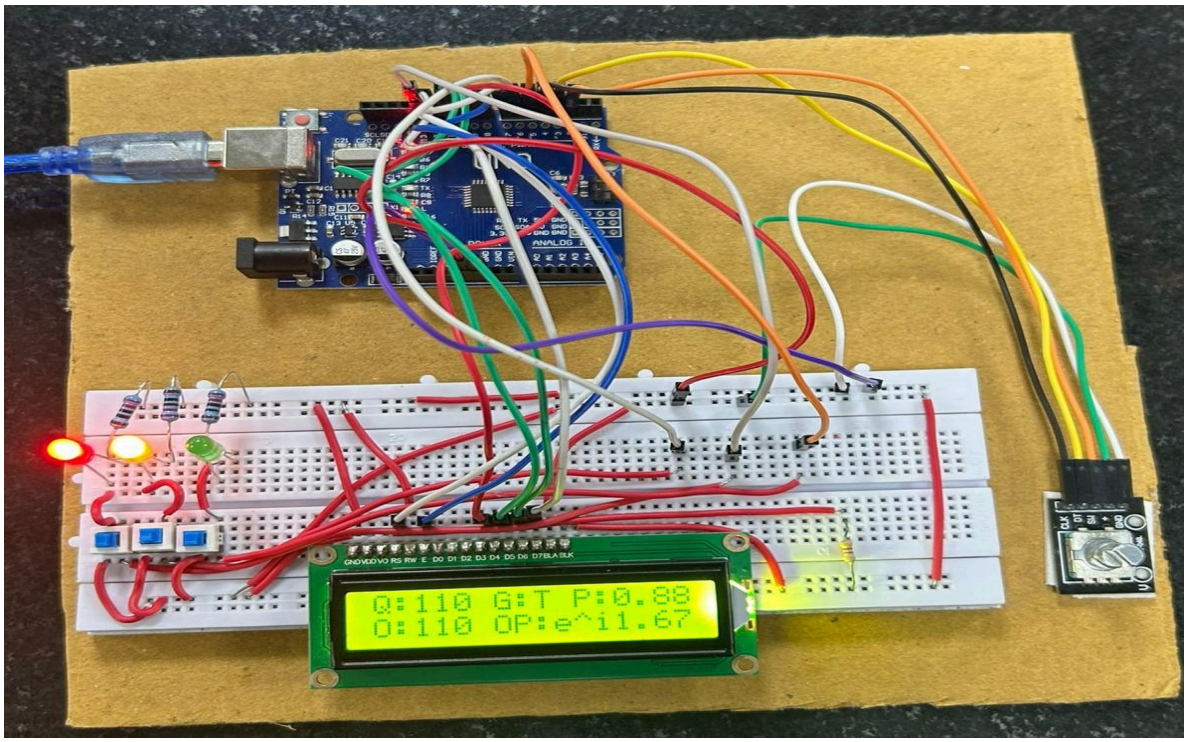
```
    case 1: // Y Gate (Simulate as a bitwise NOT with phase shift approximation)
      result ^= 0b111;
      break;

    case 2: // Z Gate (Apply phase to all bits, but keep classical bits unchanged)
      break;

    case 3: // H Gate (Hadamard approximation)
      result = ((~q1) << 2) | ((~q2) << 1) | (~q3); // Flip all bits
      break;

    case 4: // S Gate (Simulate phase gate, no classical bit changes)
      //added_phase = 1.57;
      break;

    case 5: // T Gate (Similar to S but with different phase shift)
      //added_phase = 0.79;
      break;
  }

  return result;  // Return the modified 3-bit result
}

// Main loop
void loop() {
```

```
  // Read manual input from switches
int switchState1=0;
int switchState2=0;
int switchState3=0;
  switchState1 = digitalRead(switchPin1); // Read switch state
  if (switchState1 == LOW) {            // LOW if pressed with pull-up
    qubit1 = 0;
  } else {
    qubit1 = 1;
  }
  switchState2 = digitalRead(switchPin2); // Read switch state
  if (switchState2 == LOW) {            // LOW if pressed with pull-up
    qubit2 = 0;
  } else {
    qubit2 = 1;
  }
  switchState3 = digitalRead(switchPin3); // Read switch state
  if (switchState3 == LOW) {            // LOW if pressed with pull-up
    qubit3 = 0;
  } else {
    qubit3 = 1;
  }

  // Update rotary encoder and calculate position
  encoder.tick();
```

```
  long newPosition = encoder.getPosition();
  if (newPosition != selectedGate) {
    selectedGate = newPosition % 6;  // Only 6 gates (0 to 5)
    if (selectedGate < 0) selectedGate += 6;  // Handle negative positions
  }

  // If rotary encoder button is pressed, generate a new random phase
  if (digitalRead(buttonPin) == LOW) {
    randomPhase = generateRandomPhase();
    delay(500);  // Debounce delay
  }

  // Apply the selected gate and phase to the 3-bit input
  int result = applyGate(selectedGate, qubit1, qubit2, qubit3, randomPhase);

  // Sequentially display inputs, selected gate, phase, and output on the LCD
  displayLCDSection(qubit1, qubit2, qubit3, gates[selectedGate], randomPhase, result);

  // Small delay for stability
  delay(200);
}
```

Sample Output pictures





Nikhil Sharma

Accuracy, Precision, and Error Analysis

Accuracy and Precision in the Arduino Quantum Gate Project

In our Arduino project on quantum gates, accuracy and precision were essential to achieving correct and consistent results. **Accuracy** ensured that the LED output precisely represented the expected transformations for each quantum gate selected, matching theoretical values and operations of gates like X, S, and T. By accurately mapping inputs to outputs, we could visualize gate operations reliably. **Precision** guaranteed that each run of the code produced consistent results for identical inputs, which is crucial in quantum simulations where small deviations can affect the output significantly. The project performed smoothly, with all codes running without error, reflecting high accuracy in logic implementation and precision in the hardware responses.

Sources of Error in Reproduction

When reproducing this project, several factors could introduce errors. Hardware variations, such as slight differences in switch or resistor tolerances, may lead to inconsistent input readings, affecting accuracy. The LED brightness or wiring resistance could vary, causing visibility issues or delays in output. Environmental factors like temperature fluctuations could also impact Arduino's processing, slightly delaying the expected response and thus reducing precision. Additionally, even small modifications in code structure could introduce delays or rounding errors in the quantum gate calculations, leading to a divergence from the expected output. Proper calibration and a stable power supply are essential to minimize these potential sources of error.

Troubleshooting

We faced multiple issues while developing the circuit and worked extensively to understand the problem, troubleshoot all possible fallacies, and finally solve them. We solved each problem either by trial method or with proper detailed analysis by using Google, GpT, and other such resources for help.

A near-exhaustive list of issues we faced during the development of this project is noted below, followed by their troubleshooting process.

1. The LCD display only showed a bright backlight and nothing else, the contrast was set poorly. We tried reconnecting the wires, and the VO component of the LCD to the ground, but we later found out with trial that the screen was drawing some extra current which needed to be grounded with some extra resistance, so we added a 2 kOhm resistance in series with the VO, following it to the ground, so we can dump that extra current and the contrast of the display can be set properly. It immediately showed us promising results and we were able to feed proper display into the screen.
2. LEDs were not glowing as they needed to, so we thought there must be some issue with the way we connected the LED the switch, and the digital pin. We were dipping the anode with the resistance, that was going into the ground, which created an issue so we switched it to the right position and the electrical circuit was complete thus making them glow, as the right leg was in the 5V.
3. SPDT switch wasn't making the right switching condition as the LED didn't glow as we demanded it to, and the corresponding qubit state did not change as we asked it to. We found out that the switch has three legs, at different potentials and each leg is to be used for a different purpose. We used one for connecting to the LED, one for the 5V, and the last for the Digital pin which takes the input with the corresponding high and low input.
4. We did not initially know that the LCD display has a particular initialization for the index of each position. The corresponding outputs were not displaying in the right

order or were cut off from the display making our output overflow and not showing properly. We corrected the code for the right display and the right indexing thus making our output look neat and fit perfectly into the screen.

5. There was a logical error in our interpretation of the gates, in which the phase that we added wasn't reflected in the total output phase. We re-read our basics of quantum gates and added the right code for adding the phase which displayed perfectly the final output phase. The T and S gates are now adding the corresponding phase, and the Y and Z gates are flipping the phase into negative based on the first qubit input.
6. There was a very intricate error in the way we set up the code. We were unable to figure out the error but with persistence, we found out that there was a data type error in the way we defined our functions. The function input and output were of different typesets so we changed the datatypes and thus the code was able to upload onto the Arduino.
7. There was a logical error in the way we set up the code, some of the combinations of input were giving a bit flip through code as an output, so we were not receiving a 3-qubit output, we received a binary output. However, we fixed it by displaying the first one or two qubits as significant figures every time the code ran into this loophole. There was a manual intervention needed for this issue, and we fixed it accordingly.
8. Overall, there were an umpteen number of small errors which were fixed with small hits and trials, as well as extreme persistence from all group members, which resulted in a beautiful project together.

Conclusion and Foreword

Overall, we are very excited to have done a project in under 2 months, after having learned and worked on various skills, including Programming in C, Arduino, Electronics, and some basics of the theoretical background in Quantum Gates. It was an exciting journey of learning, reading, searching, troubleshooting, as well as working in unison to bring together a great project that we can all be proud of.

In conclusion, we are able to show what we want to achieve and display something that is not available on the internet at all. We have shown a good procedure to enhance the learning in students for quantum gates and make it interactive for students of quantum technology and physics as well as mathematics. The entire project has left us with a lot of learning as well as taught us how to deal with real-time problems as well as build something that we promised. What is promised must be delivered, and we stood the test of time, thus displaying resilience and the importance of discipline among ourselves.

For a foreword note, we believe many such extensive devices can be made that enhance students' learning and better help them learn concepts in quantum computing, in a very fun and interactive manner, without having to worry about diving into the theory all of a sudden. This can help build that foundational introduction in children of younger age in schools and incline them towards this field. This can, as we called it in the introduction, become a toy that very delicately displays a crucial and intricate concept with ease for everyone to learn and help incline children towards research and development in this field, for humanity's greater good.

This project can also be further enhanced by utilizing a few techniques to make it more interactive and pleasing to the eyes while making the learning effective. We can also write an extensive MATLAB code that entails the entire state vector on a Bloch Sphere(a spherical representation of a vector in 3-D space). When each gate is pressed, it rotates the vector along the sphere using the corresponding gate. We can also improve and make the system more precise and natural by mitigating the superposition between various qubits and utilizing other gates to give a superposition

output that fluctuates as we observe, which can be called a miniature quantum computer in loose terms.

All in all, we are happy with the way our project has turned out and I would like to thank the instructor, Dr. Shouvik Dutta, along with the financial help from the university, as well the group members for showing persistence, resilience, and hard work with utmost discipline to bring this brainchild into reality. We are grateful for the opportunity and the learnings we are taking away from this project and the course will be indispensable for our future ahead.