**INTRODUCTION**

Full Adders are used to perform binary arithmetic. As with basic everyday arithmetic, they are a foundational building block and are present in almost all modern devices, such as Arithmetic Logic Units, Central Processing Units and Digital Signal Processors. The working principle behind Full Adders is that they handle carry-in and carry-out values correctly when performing mathematical operations to compute large sums which would be impractical and error-prone if done manually. The focus of the project was to construct a 2-Bit FA using discrete components and Resistor-Transistor Logic to display the output as an array of LEDs. This approach was chosen over the use of Integrated Circuits, Digital Logic or prebuilt digital logic gates to gain a better understanding of circuit building and first principles electronics in action. The report will cover the theory and construction of FAs from Logic Gates. The report covers a first-hand account of the design, simulation, testing and troubleshooting of the process. Unlike a formal or well-established construction guide, the report captures the learning curve and personal practical insights through a trial-and-error approach.

**AIM**

The primary objective of the project was to use discrete components to construct logic gates to correctly display the binary output based on the user input. The system was required to take two binary value inputs and display the binary output value using LEDs. Secondary aims of the project were to expand circuit construction knowledge as well as applying the theory of logic gate construction from university lectures through manually creating logic gates and wiring them together. In the context of non-digital circuit construction, discrete components refer to single-function elements such as resistors, diodes, capacitors and transistors, unlike integrated circuits that can perform several operations within a single package.

**BACKGROUND AND THEORY**

In basic decimal arithmetic, when the sum of two digits exceeds 9, the excess is carried over to the next column. For example, adding 7 and 3 results in 10: the units place becomes 0, and 1 is carried into the tens. Binary addition follows the same principle, except the maximum binary digit is 1. When 1 + 1 occurs, the sum resets to 0, and a carry-out of 1 is generated. Carry-in and carry-out logic is therefore essential to correctly perform binary arithmetic.

A Full Adder is an extension of the Half Adder, which is the foundational circuit capable of summing two single-bit binary inputs to produce a sum and a carry-out. However, the Half Adder cannot handle carry-in values from a previous stage, limiting its use in multi-bit arithmetic and scalable systems. The Full Adder overcomes this by introducing a third input, Carry-In (Cin), enabling the chaining of multiple adders to handle larger binary values.

The Boolean expressions for a Full Adder are:

* Sum (S) = A ⊕ B ⊕ Cin
* Carry-Out (C) = (A ∧ B) ∨ (Cin ∧ (A ⊕ B))

These functions are typically implemented using combinations of fundamental logic gates such as AND, OR, XOR, and NOT. Depending on system requirements, logic may also be constructed entirely from universal gates such as NAND or NOR. All these gates can be physically implemented using discrete components via Resistor-Transistor Logic (RTL).

**CIRCUIT CONSTRUCTION**

The construction of the 2-bit Full Adder was approached using Resistor-Transistor Logic (RTL), implemented entirely with discrete components. The logic gates required—namely XOR, AND, and OR—were first conceptualised based on ideal gate-level schematics. However, practical considerations led to a revised approach using only NAND gates, which require fewer transistors and simplify the overall layout.

*SCHEMATIC AND LOGIC DESIGN*

The theoretical design of the Full Adder includes two XOR gates, two AND gates, and one OR gate per bit. However, the XOR gate in particular requires a minimum of six transistors when implemented in RTL, whereas NAND gates require only two. Consequently, the final design leveraged universal NAND logic to reduce component count and complexity. Logic gate outputs were planned to drive LEDs, enabling visual verification of each stage during testing.

*SIMULATION AND VERIFICATION*

Prior to physical assembly, circuit behaviour was simulated using the Falstad Circuit Simulator. This allowed verification of logical functionality, identification of signal propagation paths, and preliminary testing of voltage drops across the logic stages. This simulation phase proved essential for bridging gaps between theoretical expectations and practical performance.

*BREADBOARD IMPLEMENTATION*

The circuit was assembled on a full-sized breadboard with the following inputs:

* **A** and **B**: two binary input bits
* **Cin**: carry-in

Outputs were routed through colour-coded LEDs for clarity:

* **Sum outputs**: Blue
* **Carry signals**: Red, Yellow, Green depending on stage

During initial testing, a 3.3V supply from a Raspberry Pi was used. However, the voltage drop across the LEDs (~2V–2.2V per LED) left insufficient voltage to drive the rest of the circuit. Switching to the 5V rail improved performance slightly, but excessive current draw—due to ~50 transistors and several resistors—led to overheating and partial melting of the breadboard.

*POWER MANAGEMENT AND OPTIMISATION*

A 9V battery-powered supply unit was ultimately used, resolving the power issues. Additional optimisations included:

* Removing unnecessary resistors at transistor bases after confirming safe biasing conditions
* Reducing the number of active LEDs during early-stage testing to decrease current load
* Consolidating NAND logic blocks to minimise wiring overhead

*SUBTRACTION LOGIC (PLANNED EXTENSION)*

The project included an initial plan to support subtraction by toggling the Most Significant Bit (MSB) using XOR-based inversion. While the subtraction switch was implemented, full logic-level inversion and correct MSB suppression were not finalised due to complexity in NOT gate implementation using discrete RTL.