10 July 2025

PROJECT 1: 2-BIT FULL ADDER

This report is a text only version of the full project on my website. As such, where images were provided and descriptions based around them, some edits have been made to be able to describe the project in a text only format. Where there is any confusion or discrepancy, it is advised to view the full post to gain an overall understanding of what is being discussed. Furthermore, the website has been created to act as a digital diary and log of the projects that I have completed or are currently working on. Some of the early builds rely heavily on well known, tested and accepted theory and practice, and therefore much of the theory discussion will be linked to a reliable source rather than be explained within the report itself, save for some rudimentary and introductory elements.

CONTENTS

Introduction 1
Aim 2
Backgroud and Theory 2
Circuit Construction 2
- Theory vs Practice in Circuit Design 2
- Circuit Components 3
- Building and Testing 3

Call to action 5
Rounding off and Afterthoughtss 6
Abbreviations 7

INTRODUCTION

Full Adders are the solution to being able to accurately perform binary arithmetic on a scale that allows modern devices to carry out complex calculations, which is important for things like Arithmetic Logic Units (ALU), Central Processing Units (CPU) and Digital Signal Processors (DSP), all of which are critical for everyday usage in digital applications. The project aimed to create a fully analogue 2-bit Full Adder utilising Resistor-Transistor Logic (RTL) and basic electronic components.

AIM

The primary aim of the project was to take the theory and knowledge of circuit building and binary arithmetic to construct a 2-Bit Full Adder / Subtractor with correct signing. The secondary aim of the project was to develop a deeper understanding and appreciation for circuit building and develop further circuit building skills.

BACKGROUD AND THEORY

To understand what a full adder is and what it does, it is best to turn our minds to something a lot more familiar to us - basic arithmetic. Take any two single digit integers who's sum is 10 or larger. For this example, we will take 7 and 3. Commonly, we stack the values above one another and perform the addition from right to left. We then add the values together and obtain our answer of 10. This time we take two 2-digit numbers, let's say 11 and 12. Following the same pattern, we stack them above one another and perform the arithmetic, carrying over the values that don't 'fit' into the answer box, and so we end up with 23. Now imagine that in trying to add these, or any, values together that we were not able to carry over. This is exactly what a Full Adder sets out to do - carry over values to correctly perform binary arithmetic. Adding numbers, or 'bits' in binary follows the same pattern as base-10 addition (where binary is base 2), and allows the bits to be correctly carried over to the next column to display the correct binary value (compared to its decimal counterpart). That is to say, when we add 7 (111 in binary) and 3 (011 in binary), we correctly display 1010.

Full Adders are the larger version of the smaller Half Adder, which is not able to use Carry-In Logic for computations.[1][2]

CIRCUIT CONSTRUCTION

Theory vs Practice in Circuit Design

Engineering concerns itself with finding solutions for problems that have not been encountered before: this can be on a global, or in my case an individual scale. Save for a few mandatory lab sessions building basic circuits, I hadn't attempted to build anything on this scale before.

---

[1] https://www.geeksforgeeks.org/digital-logic/half-adder-in-digital-logic/
[2] https://www.geeksforgeeks.org/digital-logic/full-adder-in-digital-logic/

Half and Full Adders are built as a combinational logic circuit and thus use logic gates which are built using transistors. Transistors themselves are used either as a switch or an amplifier, and in the case of this project, they were used as switches. The construction of the transistors leads to them operating under specific circumstances known as logic gates. The most basic of them are AND, OR & NOT. Half/Full Adders also utilise the XOR gate to operate correctly[3]. In theory the best way to build a Full Adder based on the fewest amount of logic gates is by using 2 XOR gate, 2 AND gates & an OR gate. In practice, the easiest way to build it is by using just NAND gates. **Why?** Because NAND gates require just 2 transistors, whereas building an effective XOR gate requires a minimum of 6. We therefore actually require less parts to make the Full Adder with just NAND's than with "ideal" circuit construction[4].

Circuit Components

RTL requires very few components, but some good practice techniques were employed, which will be explained later on with the build description. Parts used for this project were:

- *Power Unit*
- *9V Battery*
- *Half Sized Breadboard*
- *Full Sized Breadboard*
- *S8050 BJT NPN Transistors*
- *LED's (Various Colours)*
- *Push Buttons*
- *Wire (Various Lengths)*
- *Toggle Switch*
- *1k Ohm Resistors*
- *2.2k Ohm Resistors*

Building and Testing

Admittedly, this took a lot longer than I would have liked it to. Firstly, there was the part where I tried to brute force the build using the "ideal" schematic, which ultimately came to be a failure. During the ‘ideal’ construction, LED’s were utilized to monitor the output of each gate. This is what I was referring to when I mentioned best practice techniques. You don't necessarily need to have LED's at all points, but it's useful to implement during the build to catch anything that might be incorrect or broken along the way. Ultimately, this method

[3] https://www.geeksforgeeks.org/physics/logic-gates/
[4] https://www.gsnetwork.com/full-adder/

turned out not to work, due to how current travels through the circuit and needing extra transistors in the mix that didn't quite make sense to me, so the intuition to make adjustments and alterations wasn't there.

The breadboard consisted of 3 buttons to act as the inputs A, B, & Cin to the Full Adder, with the different LED'S used to indicate which outputs were high or low:

- *XOR1 from A or B: Blue*
- *Carry In: Yellow*
- *A & B: Green*
- *AND2 (Cin*[XOR(A*B)]): Red*

The main issue that I had encountered was with getting the final OR to function correctly. After doing some research, it seems like there were some additional transistors required to make this work, but again, this construction made it difficult for me to intuitively see how this was supposed to happen.

I turned to using Falstad to simulate the ideal and NAND style circuits, mainly to get a better understanding of the current flow and how I would relate this to building it on the breadboard itself. It's here that I started to see a lot of progress, and in the end, combining this with the GSN schematic, I managed to build 90% of the circuit in about 30 minutes. It's here that I should therefore stress the importance of not just diving into things and making sure you have a solid plan before attempting anything. There is a method to follow for a reason. A schematic can help to quickly draft out the rough idea for a circuit - what parts do we need? How many of them do we need? What do they need to do? How does that affect other parts of the circuit? And so on. The simulation is then the next best way to get the idea working and understand how to optimise what it is that you have in mind (optimisation is so important. I think I have about 3 of 4 modules across the next few years based on this principle within different areas, including a mandatory module scarily titled "Mathematical Optimisation"). My Dad used to remind me of the 5 P's:

***P**rior **P**reperation **P**revents **P**oor **P**erformance*

And with that said, the prep that I was supposed to do at the beginning now completed 2 weeks later, I was able to create a nearly fully functional 2-Bit Full Adder.

Let's take a bit of a closer look at the way this was built and constructed. I initially used the Raspberry Pi as my power source and simply run 3.3 volts to the power rails of the breadboard and to the ground of the Pico. After doing some further research about certain components within the circuit mainly the LED's, I found that there was quite a large voltage drop across them compared to what was being supplied to the circuit itself. It does depend on the colour of them, but on average you're talking between 2V – 2.2V drop across them. This

would leave virtually no voltage to go through the rest of the circuit. With that said I eventually swapped to using the 5V power supply from the Pico where I then encountered my second problem which was due in part to over engineering the circuit with too many resistors into the bases of the transistors, but I was drawing too much current from the and started melting the breadboard at one point. Now if we think about this purely from a first principles perspective of using Ohm's Law, we might be wondering why we were drawing a lot of current despite there being a low voltage source. There is only a specific amount of voltage going into the circuit, but each component will have a certain internal resistance if it is not in and of itself a resistor as well as a specific potential difference across it as well which would dictate the current that flows through that component. So given that I had around 50 transistors in the circuit as well as L eds and wires and basically every other component that requires voltage and has a resistance the amount of current that the circuit requires to work is a lot higher than what the power source can provide.

Eventually I swapped out the Raspberry Pi for a power supply unit and attached a 9 Volt battery which solved the issue almost instantly. I also removed some resistors which reduced the power dissipation across certain components as well as removing some of the initial LED's that were in the circuit.

One feature that I wanted to include was the ability to subtract and correctly show the most significant bit (MSB, the left / bottom-most LED). That means if we had a case where the answer was 0, then the LED's would be off / off / off. However, without the use of a NOT gate, if the subtraction switch was on, the MSB would be on as well, which would mean that even if the answer was 0, the LED's would be on / off / off. The logic is that if the subtraction is on, then we invert the MSB

CALL TO ACTION

Going forward I would like to make a couple of additions to this to increase the complexity and capabilities of the Adder – which essentially will become a small, integer limited calculator:

- Add a third or fourth bit
- Include the ability to multiply by 2
- Include the ability to divide by 2
- Include the ability to show decimal values

How complex do I think this will be and what sort of timeframe should I expect vs how long I will take?

Add a third or fourth bit: I have the blueprint for the Full Adder itself now, so it is just a copy and past situation which should take no more than 30 minutes

Include the ability to multiply by 2: to multiply by 2 in binary, it is a case of shifting the value to the left, so for example 001 becomes 010, 011 becomes 110 and so on. I would hopefully expect this to take not too long. With some preliminary simulations and smaller tests, I would ideally like to take only an hour

Include the ability to divide by 2: This is the inverse of the above, where bits are shifted to the right, so 110 becomes 011, 010 becomes 001 and so forth. If I can solve the multiplication, then division should be relatively simple as well and I would already have a blueprint for it, so I'll be conservative and estimate 30 minutes

Include the ability to show decimal values: This isn't something that I have been taught yet, and seeing as I would want to try and make these additions come the new semester, it would involve a bit more of a deeper dive. I would give myself 1.5 to 2 hours for the research, simulation, small scale construction and implementation.

Altogether, that comes to 4 hours in the worst-case scenario. I will keep a log of this come the time to start adding these elements in.

ROUNDING OFF AND AFTERTHOUGHTSS

Regardless of the size of any project you should always be able to take away something from it no matter how invaluable it may seem and for me this project was definitely an eye opener to what I should expect going forward. It can sometimes be hard to know how to start a project which is where the learning curve comes from in the first place

I think the biggest thing I have learnt from this is how to destroy the largest number of transistors in the shortest amount of time possible. There are some other handy things I have learned:

- Make sure that you have everything that you need before you start the project. The requirement for me to purchase more batteries transistors set me back a few days. For a personal project like this it doesn't really matter too much but there may be instances in the future where I do have a time constraint and should make sure that I'm prepared beforehand
- No matter how much you think you know about project, make sure to do some research beforehand. Information, parts & best practises can change, and when you're new to doing things it's always best to make sure you're as familiar as possible

These are learning points on a more personal level whereas the technical knowledge and understanding has a few other points. One thing that particularly surprised me and made me curious was the absence of resistors into the base of every single transistor. The global science

network schematics and construction showed that it was possible to wire the output of one part of a NAND straight to the base of another transistor. My general understanding was that resistors are required at the bases of transistors to limit the current and not overload them, however after some research it seems that if the transistor output is saturated then the voltage and current would be low enough to not overload the next stage. Generally this isn't a good idea but for small and low-level circuitry it's completely fine as long as you're not excessively overloading any part of the system.

- **Project planning:** brute force is not always the best options, unless you know exactly what to do and how to fix problems along the way. Using 'perfect' techniques like simulations can give you a good initial boost.
- **Theory vs Practice in circuit construction:** What works on paper may require adjustments and alterations in real life. There may always be another, easier way to do things
- **Intra Troubleshooting techniques:** Putting checks in place during the build to help keep you on track can save hours of work later down the line

Overall, I would say that the project was a success from the richness and the quality of the information and knowledge that I gained throughout the process. As mentioned in the beginning, my aim was to create an *analogue* Full Adder, which is always going to be a lot more difficult than just using integrated chips to complete the task, Otherwise I could have completed the project in a smaller time frame using less parts and be able to carry out much more complex arithmetic than what I am currently able to do. However, I wanted to set myself the challenge to learn along the way - that's not to say that using integrated circuits is a bad thing but I think there's something to be learned from going from first principles to complete these types of experiments.

ABBREVIATIONS

ALU – Arithmetic Logic Unit

BJT – Bipolar Junction Transistor

Cin – Carry In

CPU – Central Processing Unit

DSP – Digital Signal Processor

GSN – Global Science Network

LED – Light Emitting Diode

LSB – Least Significant Bit

MSB – Most Significant Bit

NAND – Not And

RTL – Resistor Transistor Logic

XOR – Exclusive Or