Lab 8: ad-hoc Design

**Primary Objectives:**

1. Introduce the design of complex systems using more complex components.
2. Design a 4-bit calculator using existing modules.

**Design**

I started from the functions, I decided to use a 4-bit 4-to-1 mux because the calculator had 4 functions decided by a 2-bit input.

I then worked from the first function of set data down since it seemed like an intuitive order to tackle the 4 functions.

Since the first function was to set data, I decided to use a 4-bit register since it has a “set” function and I needed a memory device for later anyways. Due to most of the functions needing the memorized value, I put the register after the mux instead of having multiple registers and making some complicated logic to have each function that required the memorized value take the correct value.

For the remaining functions I simply used the available arithmetic devices that corresponded to each function.

With all the functions implemented, I began working on how to only have GO only asserted for one operation per “cycle” of go. To do this I used a D-type flip-flop because I knew I could only have go asserted for one clock tick and would need a way to remember it. I then used a 2-bit 4-to-1 mux to only allow the value of “GO” to pass through if GO is 1 and the previous GO was 0.

**Implementation**

Figure 1: Circuit implemented from description above in design.



**Testing**

Table 1: Log of testing done on the circuit to ensure functionality.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| In | F | Go | Clock(60,280) | Out |
| 10 | 0 | 0 | 0 | 0 |
| 10 | 0 | 1 | 0 | 0 |
| 10 | 0 | 1 | 1 | 10 |
| 10 | 0 | 1 | 0 | 10 |
| 10 | 1 | 1 | 0 | 10 |
| 10 | 1 | 1 | 1 | 10 |
| 10 | 1 | 1 | 0 | 10 |
| 10 | 1 | 0 | 0 | 10 |
| 10 | 1 | 0 | 1 | 10 |
| 10 | 1 | 0 | 0 | 10 |
| 10 | 1 | 1 | 0 | 10 |
| 10 | 1 | 1 | 1 | 100 |
| 10 | 1 | 1 | 0 | 100 |
| 10 | 11 | 1 | 0 | 100 |
| 10 | 11 | 1 | 1 | 100 |
| 10 | 11 | 1 | 0 | 100 |
| 10 | 11 | 0 | 0 | 100 |
| 10 | 11 | 0 | 1 | 100 |
| 10 | 11 | 0 | 0 | 100 |
| 10 | 11 | 1 | 0 | 100 |
| 10 | 11 | 1 | 1 | 1000 |
| 10 | 11 | 1 | 0 | 1000 |
| 10 | 10 | 1 | 0 | 1000 |
| 10 | 10 | 1 | 1 | 1000 |
| 10 | 10 | 1 | 0 | 1000 |
| 10 | 10 | 0 | 0 | 1000 |
| 10 | 10 | 0 | 1 | 1000 |
| 10 | 10 | 0 | 0 | 1000 |
| 10 | 10 | 1 | 0 | 1000 |
| 10 | 10 | 1 | 1 | 110 |
| 10 | 10 | 1 | 0 | 110 |
| 10 | 10 | 1 | 1 | 110 |
| 10 | 10 | 1 | 0 | 110 |
| 10 | 10 | 1 | 1 | 110 |
| 10 | 10 | 1 | 0 | 110 |
| 10 | 10 | 0 | 0 | 110 |
| 10 | 10 | 0 | 1 | 110 |
| 10 | 10 | 0 | 0 | 110 |
| 10 | 10 | 0 | 1 | 110 |
| 10 | 10 | 0 | 0 | 110 |
| 10 | 0 | 0 | 0 | 110 |
| 10 | 0 | 0 | 1 | 110 |
| 10 | 0 | 0 | 0 | 110 |
| 10 | 0 | 1 | 0 | 110 |
| 10 | 0 | 1 | 1 | 10 |
| 10 | 0 | 1 | 0 | 10 |
| 10 | 0 | 1 | 1 | 10 |
| 10 | 0 | 1 | 0 | 10 |
| 10 | 0 | 0 | 0 | 10 |
| 10 | 0 | 0 | 1 | 10 |
| 10 | 0 | 0 | 0 | 10 |
| 10 | 1 | 0 | 0 | 10 |
| 10 | 1 | 0 | 1 | 10 |
| 10 | 1 | 0 | 0 | 10 |
| 0 | 1 | 0 | 0 | 10 |
| 0 | 0 | 0 | 0 | 10 |
| 0 | 0 | 1 | 0 | 10 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 |

**Conclusion**

Upon testing, I had an issue where the subtraction was working incorrectly. It was only working incorrectly because I had wired the subtraction incorrectly by having its inputs reversed. This has taught me that I should double check my wiring if position matters, unlike with addition or multiplication.