**Caesars Cipher Design Report**

|  |  |
| --- | --- |
| **Topic:** | *Topic 3* |
| **Date:** | *February 10th, 2023* |
| **Revision:** | *1.0* |
| **Milestone Summary:** | |  |  |  | | --- | --- | --- | | **User Story / Task** | **Hours Worked** | **Hours Remaining** | | *User can enter a phrase and receive the encoded phrase* | *4* | *2* | | *Use the switches to set different key values* | *0* | *2* | | *Enter an encoded phrase, and with the right key, get the correct message decoded* | *3* | *4* | | *Display the key being used to the 7-segment display* | *0* | *2* | | *Connect the C program to GPIO pins* | *0* | *4* | | *Create the circuit in VHDL* | *0* | *4* | |  |  |  | |  |  |  | |  |  |  | |  |  |  | |  |  |  | |  |  |  | |  |  |  | |
| **GIT URL:** | *https://github.com/hydrenoid/CaesarsCipher.git* |

**Design Documentation**

**General Technical Approach:**

*The Caesars Cipher algorithm is very simple, it takes in a phrase and encodes it using a certain key, and returns an encoded version of the phrase. Then to get the correct decoded phrase you must enter the encoded phrase and the key that was used to encode it and you will get your original phrase back. It does this by taking each character in the phrase and mapping it to an integer value and, depending on the key, add a certain number to it mapping it to a different character, to get the original character back you simply subtract the number specified by the key. In this project the user will enter their phrase into the terminal running on the ARM processor, choosing whether to encode or decode. Then they will set the key by using the first four switches, the key will be displayed on the first two 7 segment displays (using a BCD decoder to display the correct numbers). Then the program will map each character in the phrase to a number, turn that number to binary, and pass the number to the FPGA where the Caesars Cipher algorithm (which will just be a parallel adder/subtractor) using 6 GPIO pins, as well as another to select whether encoding or decoding. Then the program will read the output pins from the FPGA to get the encoded value, map the new number to its new character, and build the new phrase.*

**System Design:**

*Diagram

Description automatically generated*

**Application Design:**

*Diagram

Description automatically generated*

**Digital Logic Design:**

*7-segment Decoder*

*Truth Table:*

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Decimal Digit | A | B | C | D |  | a | b | c | d | e | f | g |
| 0 | 0 | 0 | 0 | 0 |  | 1 | 1 | 1 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 |  | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 1 | 0 |  | 1 | 1 | 0 | 1 | 1 | 0 | 1 |
| 3 | 0 | 0 | 1 | 1 |  | 1 | 1 | 1 | 1 | 0 | 0 | 1 |
| 4 | 0 | 1 | 0 | 0 |  | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 5 | 0 | 1 | 0 | 1 |  | 1 | 0 | 1 | 1 | 0 | 1 | 1 |
| 6 | 0 | 1 | 1 | 0 |  | 1 | 0 | 1 | 1 | 1 | 1 | 1 |
| 7 | 0 | 1 | 1 | 1 |  | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| 8 | 1 | 0 | 0 | 0 |  | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 9 | 1 | 0 | 0 | 1 |  | 1 | 1 | 1 | 1 | 0 | 1 | 1 |

*Boolean Expressions:*

*a = A + C + BD + ~B~D*

*b = ~B + ~C~D + CD*

*c = B + ~C + D*

*d = ~B~D + C~D + B~CD + ~BC + A*

*e = ~B~D + C~D*

*f = A + ~C~D + B~C + B~D*

*g = ~BC + C~D + B~C + A*

*Caesars Cipher*

*Final Circuit Diagrams:*

*Diagram, engineering drawing

Description automatically generated*

*Diagram

Description automatically generated*

**VLSI Design:**

*N/A In development, check system design and final circuit diagrams*

**Risks and Issues:**

*Some issues that I have already delt with were 1) getting the application to set the GPIO pins with the correct binary number as many of the binary numbers were themselves smaller than 6 bits, so I had to create an algorithm to create a binary array in the correct order and 2) deciding how many pins I was going to need depending on the mapping values I was using. Originally I wanted to do ascii values and I might move back to that but I wanted to limit the number of pins that I was using to six for now so I had to create my own map for the characters and functions to map them. For risks in the future it will mostly be making sure that the circuit diagrams for the Cipher are built correctly and making sure that there are no instances where it fails. Along with that the biggest step will be finding a way to easily manipulate the GPIO pins inside of the register as that will be the majority of what the program does reading and writing to that register.*

**Other Documentation:**

*Pin Assignments:*

|  |  |
| --- | --- |
| gpio | pin |
| a0 | PIN\_W15 |
| a1 | PIN\_AK2 |
| a2 | PIN\_Y16 |
| a3 | PIN\_AK3 |
| a4 | PIN\_AJ1 |
| a5 | PIN\_AJ2 |
| b0 | PIN\_AH2 |
| b1 | PIN\_AH3 |
| b2 | PIN\_AH4 |
| b3 | PIN\_AH5 |
| b4 | PIN\_AG1 |
| b5 | PIN\_AG2 |
| d0 | PIN\_AG3 |
|  |  |
| display | pin |
| e0 | PIN\_W17 |
| e1 | PIN\_V18 |
| e2 | PIN\_AG17 |
| e3 | PIN\_AG16 |
| e4 | PIN\_AH17 |
| e5 | PIN\_AG18 |
| e6 | PIN\_AH18 |
| f0 | PIN\_AF16 |
| f1 | PIN\_V16 |
| f2 | PIN\_AE16 |
| f3 | PIN\_AD17 |
| f4 | PIN\_AE18 |
| f5 | PIN\_AE17 |
| f6 | PIN\_V17 |

|  |  |
| --- | --- |
| switches | pin |
| h0 | PIN\_AB30 |
| h1 | PIN\_Y27 |
| h2 | PIN\_AB28 |
| h3 | PIN\_AC30 |