CS132 Computer Organisation and Architecture

Coursework 1 2022/23 Solutions

Student ID: <2208321>

Completed: Wednesday, 23 November 2022

# Question 1 – Boolean Logic

## Part A

Using the following equation (where X is an output and A, B, C and D are inputs):

### Section i [5]

Simplify the original equation using boolean algebra. Show all the rules you applied when doing this.

|  |
| --- |
| **Solution:** De Morgan  Absorption  ()  Absorption () |

### Section ii [5]

Simplify the original equation using Karnaugh Maps (K-Maps). Assume that all the outputs need to be exactly correct (i.e. no “Don’t Care" outputs).

|  |
| --- |
| **Solution:** A picture containing chart  Description automatically generated |

### Section iii [4]

Design a circuit using either of the simplified equation.

|  |
| --- |
| **Solution: A piece of paper with numbers and letters on it  Description automatically generated with low confidence** |

## Part B [4]

Using NAND gates, build a NOR gate.

|  |
| --- |
| **Solution: 4 nand gates** |

## Part C [7]

Using NOR gates, build a NAND gate.

|  |
| --- |
| **Solution: 4 Nor gates Diagram  Description automatically generated** |

# Question 2 - Computational logic

## Part A

In both the lectures and the labs, we touched on 1-bit full adders, but did not explicitly specify how one could be designed.

### Section i [9]

Design a 1-bit full adder using as few logic gates as possible. Inputs A, B and Carryin, alongside outputs Sum and Carryout; should be clearly labelled.

### Section ii [4]

For each logic gate, label the inputs and outputs and provide a truth table.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Solution:**   | x | y | carry in | carry out | sum | | --- | --- | --- | --- | --- | | 0 | 0 | 0 | 0 | 0 | | 0 | 0 | 1 | 0 | 1 | | 0 | 1 | 0 | 0 | 1 | | 0 | 1 | 1 | 1 | 0 | | 1 | 0 | 0 | 0 | 1 | | 1 | 0 | 1 | 1 | 0 | | 1 | 1 | 0 | 1 | 0 | | 1 | 1 | 1 | 1 | 1 |  |  | 00 (noty and not carry in) | 01(not y and carry in) | 11 (y and carry in) | 10(y and not carry in) | | --- | --- | --- | --- | --- | | 0 (notx) |  | 1 |  | 1 | | 1 (x) | 1 |  | 1 |  |   K Map for sum  sum = x¬ycarryin + ¬xy¬carryin + x¬y¬carryin + xycarryin  = xcarryin + y +x+ xycarryin  x ⨁ y ⨁ carryin  carry out = ycarryin + xcarryin + xy +xycarryin  = (x ⨁ y)carryin +xy  only 2 inputs can be true at a time.   |  | 00 (noty and not carryin) | 01(not y and carryin) | 11 (y and carryin) | 10(y and not carryin) | | --- | --- | --- | --- | --- | | 0 (notx) |  |  | 1 |  | | 1 (x) |  | 1 | 1 | 1 | |  |  |  |  |  |   K map for carry out  **A picture containing text, whiteboard  Description automatically generated**  **ii)**   |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | | **A** | **B** | **Carry in** | **A ⨁ B**  **Inputs= a and b** | **A ⋀ B**  **Inputs= a and b** | **Sum**  **Inputs= (A ⨁ B) and Carry in**  **Output=**  **(A ⨁ B) ⨁ Carryin** | **Inputs = (A ⨁ B) and Carry in**  **Output= (A ⨁ B) ⋀ Carry in** | **Carry out**  **Inputs = ((A ⨁ B) ⋀ Carry in)** and (A**⋀B)**  **Output = ((A ⨁ B) ⋀ Carry in)** ⋁ (A**⋀B)** | | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | | **0** | **0** | **1** | **1** | **0** | **1** | **1** | **0** | | **0** | **1** | **0** | **1** | **0** | **1** | **0** | **0** | | **0** | **1** | **1** | **0** | **0** | **0** | **0** | **1** | | **1** | **0** | **0** | **1** | **0** | **1** | **0** | **0** | | **1** | **0** | **1** | **0** | **0** | **0** | **0** | **1** | | **1** | **1** | **0** | **0** | **1** | **0** | **0** | **1** | | **1** | **1** | **1** | **0** | **1** | **1** | **0** | **1** | |

## Part B

In both the lectures and the labs, we also covered decoders amongst others. Consider an active-low 3 × 8 decoder.

### Section i [9]

Design an active-low 3 × 8 decoder using only NOR gates. You should clearly label any inputs and outputs.

|  |
| --- |
| **Solution:Diagram  Description automatically generated**  **Shape, arrow  Description automatically generated** |

### Section ii [3]

Provide a truth table for an active-low 3 × 8 decoder.

|  |
| --- |
| **Solution:A picture containing letter  Description automatically generated** |

# Question 3 - Sequential logic

## Part A

In lectures, we explored an N-bit Counter, where the value decreased with each clock cycle.

### Section i [12]

Design a 4-bit bi-directional counter (where the value can count up or down, depending on an input signal). All inputs and outputs should be clearly labelled.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Solution:**  Td, Tc, Tb, Ta are the inputs. Qd, Qc, Qb, Qa are the outputs of flip flops and are the outputs of the 4 bit counter.  Control Mod/control input known as M. And depending on M’s value we will upwards or downwards. In this case, I have **assumed** that when M is 0 the counter counts upwards and when M is 1, the counter counts downwards.   | **M** | **Qd Qc Qb Qa** | **Qd+ Qc+ Qb+ Qa+** | **Td Tc Tb Ta** | | --- | --- | --- | --- | | 0 | 0 0 0 0 | 0 0 0 1 | 0 0 0 1 | | 0 | 0 0 0 1 | 0 0 1 0 | 0 0 1 1 | | 0 | 0 0 1 0 | 0 0 1 1 | 0 0 0 1 | | 0 | 0 0 1 1 | 0 1 0 0 | 0 1 1 1 | | 0 | 0 1 0 0 | 0 1 0 1 | 0 0 0 1 | | 0 | 0 1 0 1 | 0 1 1 0 | 0 0 1 1 | | 0 | 0 1 1 0 | 0 1 1 1 | 0 0 0 1 | | 0 | 0 1 1 1 | 1 0 0 0 | 1 1 1 1 | | 0 | 1 0 0 0 | 1 0 0 1 | 0 0 0 1 | | 0 | 1 0 0 1 | 1 0 1 0 | 0 0 1 1 | | 0 | 1 0 1 0 | 1 0 1 1 | 0 0 0 1 | | 0 | 1 0 1 1 | 1 1 0 0 | 0 1 1 1 | | 0 | 1 1 0 0 | 1 1 0 1 | 0 0 0 1 | | 0 | 1 1 0 1 | 1 1 1 0 | 0 0 1 1 | | 0 | 1 1 1 0 | 1 1 1 1 | 0 0 0 1 | | 0 | 1 1 1 1 | 0 0 0 0 | 1 1 1 1 | | 1 | 0 0 0 0 | 1 1 1 1 | 1 1 1 1 | | 1 | 0 0 0 1 | 0 0 0 0 | 0 0 0 1 | | 1 | 0 0 1 0 | 0 0 0 1 | 0 0 1 1 | | 1 | 0 0 1 1 | 0 0 1 0 | 0 0 0 1 | | 1 | 0 1 0 0 | 0 0 1 1 | 0 1 1 1 | | 1 | 0 1 0 1 | 0 1 0 0 | 0 0 0 1 | | 1 | 0 1 1 0 | 0 1 0 1 | 0 0 1 1 | | 1 | 0 1 1 1 | 0 1 1 0 | 0 0 0 1 | | 1 | 1 0 0 0 | 0 1 1 1 | 1 1 1 1 | | 1 | 1 0 0 1 | 1 0 0 0 | 0 0 0 1 | | 1 | 1 0 1 0 | 1 0 0 1 | 0 0 1 1 | | 1 | 1 0 1 1 | 1 0 1 0 | 0 0 0 1 | | 1 | 1 1 0 0 | 1 0 1 1 | 0 1 1 1 | | 1 | 1 1 0 1 | 1 1 0 0 | 0 0 0 1 | | 1 | 1 1 1 0 | 1 1 0 1 | 0 0 1 1 | | 1 | 1 1 1 1 | 1 1 1 0 | 0 0 0 1 |   A picture containing timeline  Description automatically generated  Excitation table for T type flip flop-   | **Qn** | **Qn+1** | **T** | | --- | --- | --- | | 0 | 0 | 0 | | 0 | 1 | 1 | | 1 | 0 | 1 | | 1 | 1 | 0 |   Karnaugh Maps for each flip flop-  A piece of paper with writing on it  Description automatically generated  Solving the K Maps produces the following expression for each T flip flop.  Text, letter  Description automatically generated  **Diagram  Description automatically generated** |

### Section ii [5]

Explain how this counter has been designed and any assumptions that have been made.

|  |
| --- |
| 1. **Since it is a 4 bit counter, 4 T type flip flops are used.** 2. **A control input, M, is used. If M = 0 it works as a up counter. If M=1, it works as a down counter.** 3. **Table is drawn, with M, current state of Qd, Qc, Qb and Qa, next state of Qd, Qc, Qb and Qa and Td, Tc, Tb and Ta.** 4. **Using the excitation table, Td, Tc, Tn and Ta are filled.** 5. **Their K maps are drawn and are implemented using logic gates.** 6. **An assumption is that modelled logic gates are acting instantly (that there is no propagation delay). Change in clock signal doesn’t instantly changes M, Qa, Qb, Qc and Qd. Logic gates have propagation delay and the change in input might not be registered by the output for a relatively significant amount of time even after the clock pulse changing.** |

## Part B

In lectures, we also explored the use of 3-state logic.

### Section i [4]

Draw a truth table for a 3-state buffer, with the inputs In and Enable, and the output Out. Assume that Enable is active-high.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Solution:**   |  |  |  | | --- | --- | --- | | **In** | **Enable** | **Out** | | **0** | **0** | **Z** | | **0** | **1** | **0** | | **1** | **0** | **Z** | | **1** | **1** | **1** |   **In = Out when enable = 1.**  **Z- High Impedance which means very little current passes through the circuit.** |

### Section ii [4]

Using as few logic gates as possible, design a circuit for the 3-state buffer as described in the previous sub-question. Clearly label any inputs and outputs.

|  |
| --- |
| **Solution:**  **Output Z is presumed to be 0 in the circuit since very little current passes through the buffer at this stage.**  **Diagram  Description automatically generated** |

# Question 4 - Assembly

## Part A [3]

Early forms of division relied on repeatedly subtracting one number from another until the result was less than that the number being substituted. Assume we have 111112 ÷01002, what would be the answer after the first iteration of the loop? Your answer should be represented as a binary Two’s Complement number (2TC).

|  |
| --- |
| **Solution: 11111 = 31(denary)**  **0100 = 4 (denary)**  **31-4 =27**  **Using 6 bits, (6 bits are needed to represent 11111 in 2TC), 31 = 011111 (2TC)**  **4 = 000100 (2TC0**  **31 – 4 ≡ 31 + (-4)**  **-4 = 111100**  **011111**  **+ 111100**  **= 011011 (2TC)** |

## Part B [14]

The algorithm shown in Algorithm 1 shows the pseudo-code for a simple divide algorithm. Write an assembly code that implements this algorithm. This code should be able to be ran in MicSim. Therefore, your implementation should use the 68K instruction set.

|  |  |
| --- | --- |
| ***Algorithm 1: Algorithm for basic divide functionality*** | |
| i ← 63 | ▷ This is the numerator, and will be the remainder at the end of the program |
| j ← 3 | ▷ This is denominator |
| k ← 0 | ▷ At the end, this will be result |
| **while** i ≥ j **do** |  |
| i←i−j |  |
| k←k+1 |  |
| **end while** |  |

|  |
| --- |
| **Solution:**  **| declare three constants**  **i: dc.b #63**  **j: dc.b #3**  **k: dc.b #0**  **|declare storage of one byte in main memory for the result of k and .**  **resultk: ds.b 1**  **move.b i, D0**  **move.b j, D1**  **move.b k, D2**  **|Main loop**  **| i-j**  **while: sub.w D1, D0**  **| if I is greater than or equal to branch to increment k, otherwise end the loop.**  **beg igreaterequal**  **jmp 0**  **igreaterequal: inc.b D2**  **move.b D2, resultk**  **jmp while**  **\*Attached 4b.asm file for the code\*** |

## Part C [5]

Show how your code will produce the correct answer by showing the values of i, j and k at the completion of each loop iteration.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Solution:**  **Each row corresponds to a loop iteration.**   |  |  |  |  | | --- | --- | --- | --- | | I | J | K | i≥j | | 63 | 3 | 0 | Y | | 60 | 3 | 1 | Y | | 57 | 3 | 2 | Y | | 54 | 3 | 3 | Y | | 51 | 3 | 4 | Y | | 48 | 3 | 5 | Y | | 45 | 3 | 6 | Y | | 42 | 3 | 7 | Y | | 39 | 3 | 8 | Y | | 36 | 3 | 9 | Y | | 33 | 3 | 10 | Y | | 30 | 3 | 11 | Y | | 27 | 3 | 12 | Y | | 24 | 3 | 13 | Y | | 21 | 3 | 14 | Y | | 18 | 3 | 15 | Y | | 15 | 3 | 16 | Y | | 12 | 3 | 17 | Y | | 9 | 3 | 18 | Y | | 6 | 3 | 19 | Y | | 3 | 3 | 20 | Y | | 0 | 3 | 21 | N | |

## Part D [3]

The current algorithm does not deal with the “Divide-by-0” error. What would happen if j = 0 at the beginning, and how could this be fixed?

|  |
| --- |
| **Solution: infinite loop as I value will always remain the same that is 63, which is higher than j. a (pre)condition can be placed at the start to check if j =0.**  **Since i-j will always be 63 (63-0=63), it will infinitely loop because I will always be higher than j (63 will always be greater than 0). A pre-condition could be placed at the start to check if j=0.**  **Since k=0, if j-k returns 0, then j must be 0. Then jump to end the program.**  **| declare three constants**  **i: dc.b #63**  **j: dc.b #0**  **k: dc.b #0**  **|declare storage of one byte in main memory for the result of k.**  **resultk: ds.b 1**  **move.b i, D0**  **move.b j, D1**  **move.b k, D2**  **|Main loop**  **| check if j =0. since k=0, if j-k=0, then j must be 0. This then ends the program since division by 0 not possible.**  **sub.b D1, D2**  **beq end**  **| i-j**  **while: sub.w D1, D0**  **| if I is greater than or equal to branch to increment k, otherwise end the loop.**  **beg igreaterequal**  **jmp 0**  **igreaterequal: inc.b D2**  **move.b D2, resultk**  **jmp while**  **end: jmp 0**  **\*Attached a2.asm file for the code\*** |