Contents

Chapter 1

Part 3 (4-bit binary to BCD converter)

1.1 Problem Statement

•Using the conditional adder modules, design a combinational 4-bit binary to BCD converter
•Test that it works by applying appropriate inputs and checking the outputs
•Label the terminals to reflect their roles
•Save it as a regular circuit (logic file), reopen and retest

1.2 Introduction

Part 3 involves us to create a 4-bit binary to BCD converter

1.3 Solution Description

1.3.1 Input:

$$(A_3A_2A_1A_0)_2$$

are the inputs provided as a 4-bit binary integer that must be translated into the matching BCD. To do this, we shall employ the double dabble algorithm.

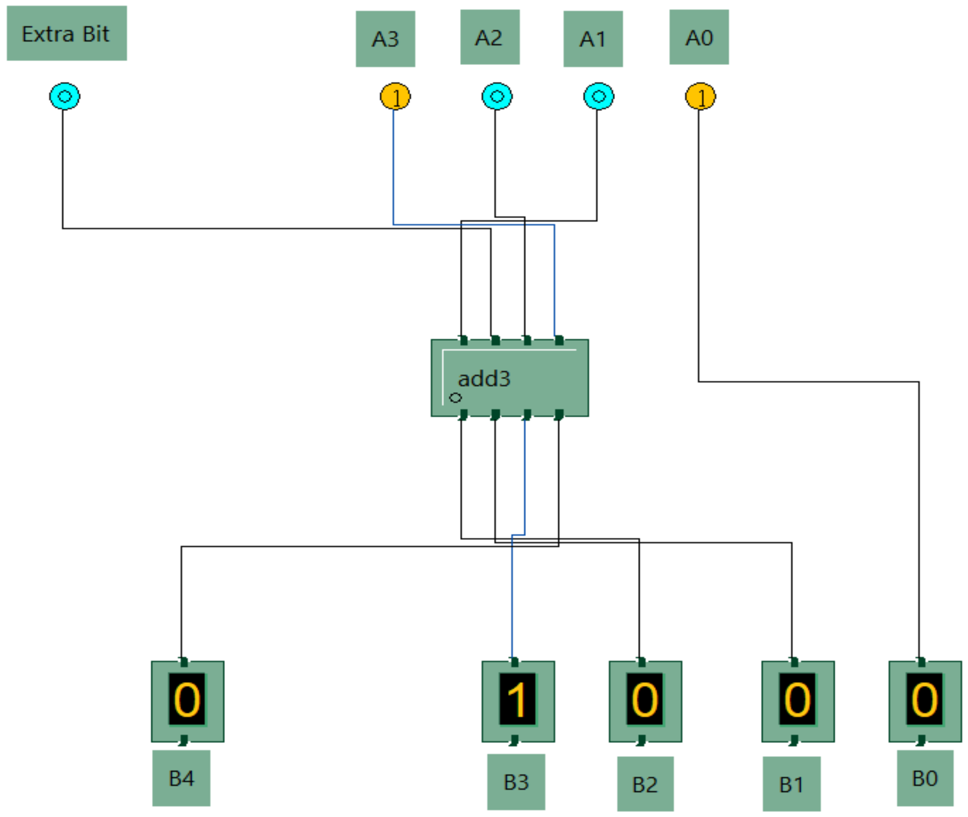
1.3.2 Circuits Description:

To implement the double dabble algorithm, we used the '≥ 5 3 adder' component produced in part two of this project. The component takes four input bits, applies a comparator (≥ 5), and adds three to the input if the result is affirmative, i.e. $(A_3A_2A_1A_0)_2 \geq (0101)_2$. In all other cases, it outputs the same four bits.

1.3.3 Algorithm Used:

1. The input is a four-bit binary number. The maximum possible input is $(1111)_2 = (15)_{10}$. As output, we must return its BCD representation.
2. To do this, we shall employ the double dabble algorithm. In this approach, we read input bit by bit by performing a left-shift in sequential increments.
3. Before completing the left-shift operation, we run each nibble through the '≥ 5 3 adder', which means add $3 = (0011)_2$ if any nibble has value greater than or equal to five.
4. Because the input is 4-bit, and we know that any number ≥ 5 in decimal form will contain ≥ 3 digits in binary form. So we'll take the first three digits from the left and see if they are ≥ 5 or not.
5. We take a dummy digit (set to 0) as the most significant bit i.e. we give $(0A_3A_2A_1)_2$ as the input to the conditional adder.
6. If it is ≥ 5, we will add 3 to it and then shift by one position.
7. For implementing a left-shift, the output of the '≥ 5 3 adder' will map to the most significant four digits of the output and then the last bit of the input will map to last bit of the output to give $(B_4B_3B_2B_1A_0)$ BCD as final result.

1.4 Digital Logic Circuits



1.5 Inference/Observations

1. The input and output have the same least significant digit.
2. After sending them through the $\geq$ 5 3 adder, we manually shift the first three digits and map them to the first four digits of the output. Because the $\geq$ 5 3 adder accepts four inputs, we use 0 as the most significant digit so that the value does not change.
3. We can expand on this concept by creating a 6-bit and 7-bit binary to BCD converter.

Chapter 2

Part 4 (6-bit binary to BCD converter)

2.1 Problem Statement

- Using the conditional adder modules, design a combinational 6-bit binary to BCD converter
- Test that it works by applying appropriate inputs and checking the outputs
- Label the terminals to reflect their roles
- Save it as a regular circuit (logic file), reopen and retest

2.2 Introduction

Part 4 involves us to write a 6-bit binary to BCD converter

2.3 Solution Description

2.3.1 Input:

$(A_5A_4A_3A_2A_1A_0)_2$ are the inputs, which are presented as a 6-bit binary numbers that must be transformed into equivalent BCD. To do this, we shall employ the double dabble algorithm.

2.3.2 Circuit Description:

To implement the double dabble algorithm, we used the '$\geq$ 5 3' adder component produced in part two of this project. The component takes four input bits, applies a comparator (≥ 5), and adds three to the output if the result is affirmative. In all other cases, it outputs the same four bits.

2.3.3 Algorithm Used:

1. The input consists of a 6-bit binary number. The maximum possible input is $(111111)_2 = (63)_{10}$. As output, we must return its BCD representation.

2. To do this, we shall employ the double dabble algorithm. In this approach, we read input, bit by bit by performing a left-shift in sequential increments.

3. Before completing the left shift operation, we run each nibble through the '$\geq$ 5 3 adder,' which means add $3 = (0011)_2$ if any nibble has value greater than or equal to five.

4. Because $(101)_2 = (5)_{10}$, we know that any number ≥ 5 in decimal form will have 3 digits in binary form. This means that we don't need to feed any nibble with less than three digits via the conditional adder.

5. So we'll take the first three digits from the left and see if they're ≥ 5 or not. If it is ≥ 5, we shall add 3 to it before shifting by one bit.

6. The most significant digit of the output BCD will be the most significant digit of the output of this '$\geq$5 3 adder'.

7. We'll use another '$\geq$5 3 adder,' with its input being the last component's remaining three outputs and the next bit in the input number. In this manner, we carry out a left shift operation.

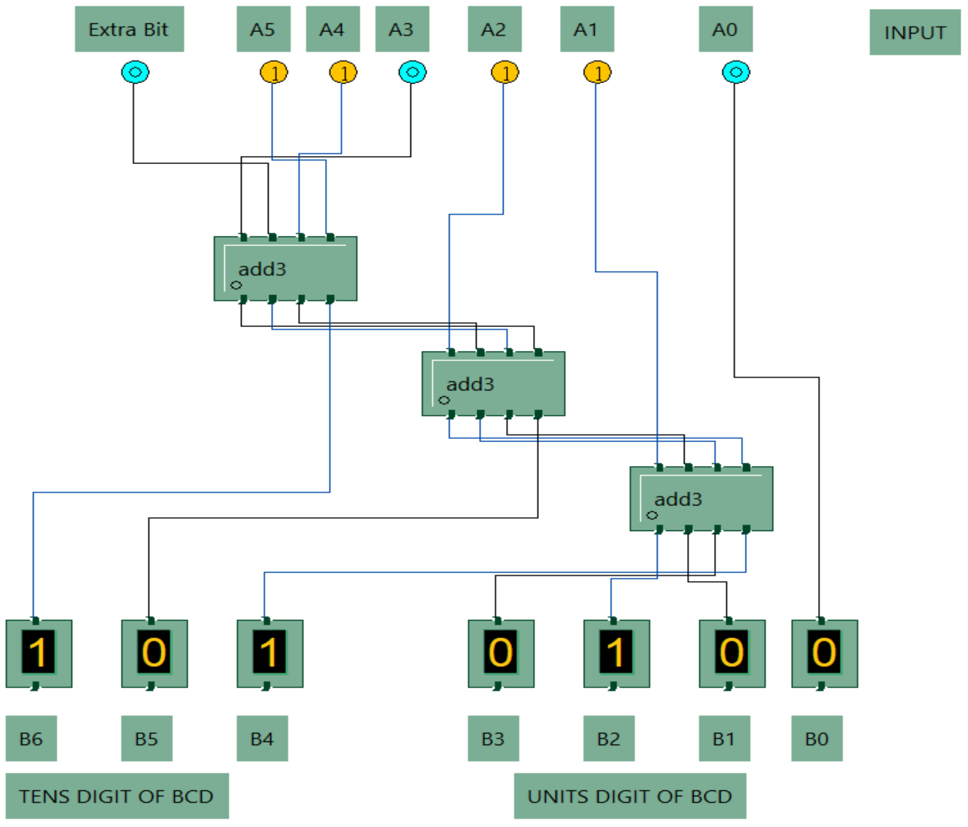
8. The outputs will be mapped in the same way, and we shall proceed. We will utilise this frequently and gradually change.

9. When we read $A_3A_2A_1$ we will evaluate them with one more '$\geq$ 5 3 adder'.

10. The output of this '≥5 3 adder' will be mapped to $B_4B_3B_2B_1$ bits in the final

11. The last bit of the input will be mapped to the last bit of output.

2.4 Digital Logic Circuits



2.5 Inference/Observations

1. The input and output have the same least significant digit.

2. Because the '≥5 3' adder requires four inputs, we provide 0 as a dummy bit which is the most significant digit.

3. We can expand on this concept by creating a 7-bit binary to BCD converter.

Chapter 3

Part 5 (7-bit binary to BCD convector)

3.1 Problem Statement

- Using the conditional adder modules, design a combinational 7-bit binary to BCD converter
- Test that it works by applying appropriate inputs and checking the outputs
- Label the terminals to reflect their roles
- Save it as a regular circuit (logic file), reopen and retest

3.2 Introduction

Part 5 involves us to write a 7-bit binary to BCD converter

3.3 Solution Description

3.3.1 Input:

$A_6A_5A_4A_3A_2A_1A_0$ The inputs are presented as 7-bit binary numbers that must be translated into equivalent BCD. To do this, we shall employ the double dabble algorithm.

3.3.2 Circuit Description:

To implement the double dabble algorithm, we used the ≥ 5 3 adder component produced in part two of this project. The component takes four input bits, applies a comparator (≥ 5), and adds three to the output if the result is positive. In all other cases, it outputs the same four bits.

3.3.3 Algorithm:

1. The input consists of a 7-bit binary number. The maximum possible input is $(1111111)_2 = (127)_{10}$. As output, we must return its BCD representation.

2. To do this, we shall employ the double dabble method. In this approach, we read input bit by bit by performing a left-shift in sequential increments.

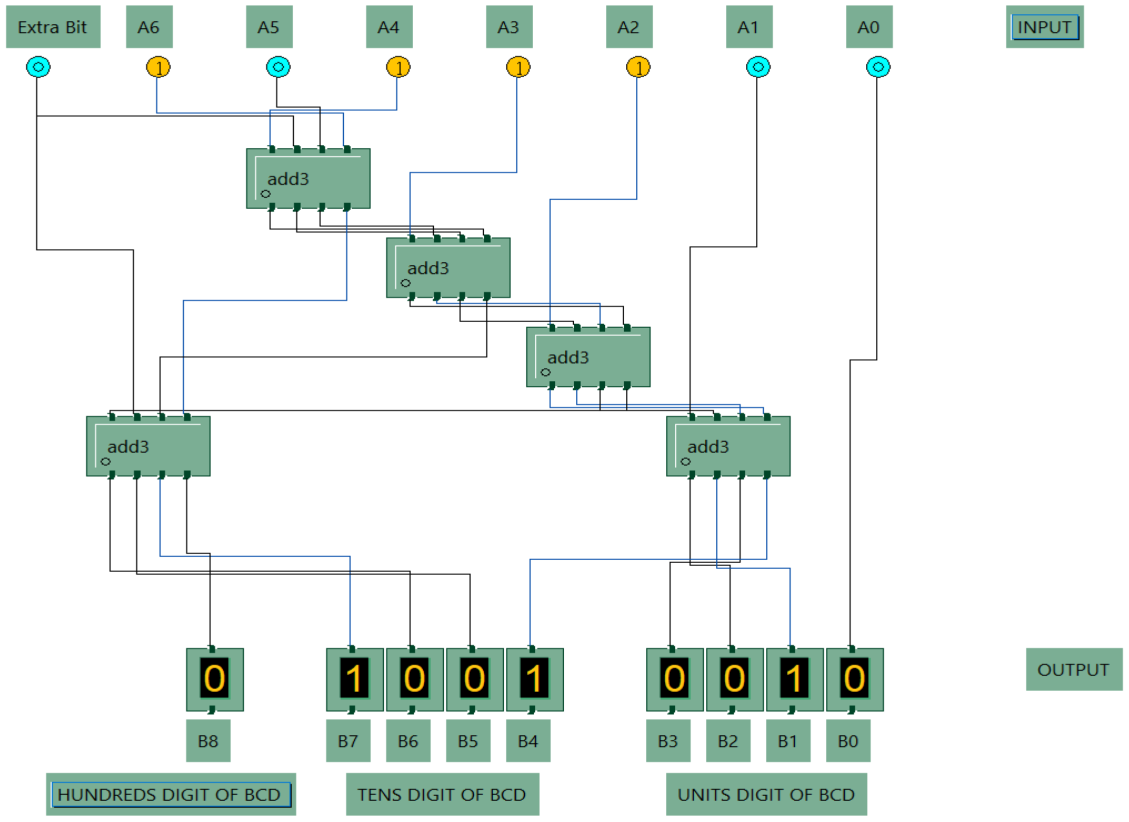
3. Prior to completing the left shift operation, we run each nibble through the '≥ 5 3 adder,' i.e. add $3 = (0011)_2$ if any nibble has value ≥ 5.

4. Because $(101)_2 = (5)_{10}$, we know that any number ≥ 5 in decimal form will have 3 digits in binary form. This means that we don't need to feed any nibble with less than three digits via the conditional adder.

5. To implement a left-shift, in the next step, the input of '≥ 5 3 adder' will be the least significant three bits of the last component and the next bit in the input number.

6. When the second nibble has enough digits (≥ 3), we will evaluate it with one more '≥ 5 3 adder (C_1)' before performing the left-shift operation for reading the next (in this case last) input bit.

3.4 Digital Logic Circuits



3.5 Inference/Observations

1. The input and output have the same least significant digit.
2. Because the 5 3 adder accepts four inputs, we provide 0 as a dummy bit which is the most significant digit for a three-bit input.
3. These binary to binary coded decimal converters can theoretically be extended to any number of bits. All we have to do is use the double dabble method to read the input bit by bit and add the qualifying nibbles.