I. 1.d Multistate Logic

A combination of n binary digits (bits), can represent 2^n unique numbers. In this case, this system represents the set $\{x \in \mathbb{Z} | 0 \leq x < 2^n\}$. In the first section, the application of a set of operations, on this system is analysed. Later, the system is expanded by a special "Undefined" token. The analysis of these 2 Systems, is then transferred to bigger combinations of bits.

A. Operations

- AND: An and Operation on 2 binary numbers, can be performed bitwise, which for any 2 n digit binary numbers, yields a n digit number as a result
- OR: Similarly to the AND operation, the OR operation can be performed bitwise, to yield a number in the range of $[0, 2^n]$
- XOR: The XOR operation of 2 bits, yields 1 if only one of the inputs is 1 and otherwise yields 0
- NOT: A NOT Operation is usually performed to incert a single bit, for a comparison of 2 numbers, the Operation is performed as NOT EQUAL
- ADD: The ADD Operation, adds the 2 numbers and truncates them to the last 3 bits, which are represented by the bit sequence
- SUB: The SUB Operation, subtracts the 2nd number from the 1st and truncates them to the last 3 bits, which are represented by the bit sequence
- MUL: The MUL Operation, multiplies the 2 numbers and truncates them to the last 3 bits, which are represented by the bit sequence
- DIV: The DIV Operation, diviedes the 1st number by the 2nd and truncates them to the last 3 bits, which are represented by the bit sequence

B. Truth Table

1	2	AND	OR	XOR	NOT	ADD	SUB	MUL	DIV
000	000	000	000	000	000	000	000	000	Divide by zero
000	001	000	001	001	001	001	111	000	000
000	010	000	010	010	001	010	110	000	000
000	011	000	011	011	001	011	101	000	000
000	100	000	100	100	001	100	100	000	000
000	101	000	101	101	001	101	011	000	000
000	110	000	110	110	001	110	010	000	000
000	111	000	111	111	001	111	001	000	000
001	000	000	001	001	001	001	001	000	Divide by zero
001	001	001	001	000	000	010	000	001	001
001	010	000	011	011	001	011	111	010	000
001	011	001	011	010	001	100	110	011	000
001	100	000	101	101	001	101	101	100	000
001	101	001	101	100	001	110	100	101	000
001	110	000	111	111	001	111	011	110	000
001	111	001	111	110	001	000	010	111	000

1	2	AND	OR	XOR	NOT	ADD	SUB	MUL	DIV
010	000	000	010	010	001	010	010	000	Divide by zero
010	001	000	011	011	001	011	001	010	010
010	010	010	010	000	000	100	000	100	001
010	011	010	011	001	001	101	111	110	000
010	100	000	110	110	001	110	110	000	000
010	101	000	111	111	001	111	101	010	000
010	110	010	110	100	001	000	100	100	000
010	111	010	111	101	001	001	011	110	000
011	000	000	011	011	001	011	011	000	Divide by zero
011	001	001	011	010	001	100	010	011	011
011	010	010	011	001	001	101	001	110	001
011	011	011	011	000	000	110	000	001	001
011	100	000	111	111	001	111	111	100	000
011	101	001	111	110	001	000	110	111	000
011	110	010	111	101	001	001	101	010	000
011	111	011	111	100	001	010	100	101	000
100	000	000	100	100	001	100	100	000	Divide by zero
100	001	000	101	101	001	101	011	100	100
100	010	000	110	110	001	110	010	000	010
100	011	000	111	111	001	111	001	100	001
100	100	100	100	000	000	000	000	000	001
100	101	100	101	001	001	001	111	100	000
100	110	100	110	010	001	010	110	000	000
100	111	100	111	011	001	011	101	100	000
101	000	000	101	101	001	101	101	000	Divide by zero
101	001	001	101	100	001	110	100	101	101
101	010	000	111	111	001	111	011	010	010
101	011	001	111	110	001	000	010	111	001
101	100	100	101	001	001	001	001	100	001
101	101	101	101	000	000	010	000	001	001
101	110	100	111	011	001	011	111	110	000
101	111	101	111	010	001	100	110	011	000
110	000	000	110	110	001	110	110	000	Divide by zero
110	001	000	111	111	001	111	101	110	110
110	010	010	110	100	001	000	100	100	011
110	011	010	111	101	001	001	011	010	010
110	100	100	110	010	001	010	010	000	001
110	101	100	111	011	001	011	001	110	001
110	110	110	110	000	000	100	000	100	001
110	111	110	111	001	001	101	111	010	000
111	000	000	111	111	001	111	111	000	Divide by zero
111	001	001	111	110	001	000	110	111	111
111	010	010	111	101	001	001	101	110	011
111	011	011	111	100	001	010	100	101	010
111	100	100	111	011	001	011	011	100	001
111	101	101	111	010	001	100	010	011	001
111	110	110	111	001	001	101	001	010	001
111	111	111	111	000	000	110	000	001	001

C. Incorret Results

An incorrect result, can appear in all of the non primary Operations, namely the ADD, SUB, MUL and DIV. These can be either backtraced to an over or underflow, in the case of ADD, SUB or MUl, or are the result of integer truncation and division by 0. An overflow, occurs when the result needs more digits than are available, namely if $x \geq 2^n$. An underflow, is the opposite of the overflow and happens when the result of the calculation is smaller than the smallest representable number, in our case $x < 0$. For a DIV operation, for every operation x/y where $x \mod y \neq 0$ the result is incorrect, as the number system does not represent fractions.

Listing the Conditions for a valid result of the non primary operations on x and y in an n bits system, yields the following List.

- ADD:
  - $x + y < 2^n$
- SUB:
  - $x \geq y$
- MUL:
  - $x * y \leq 2^n$
- DIV:
  - $x \mod y = 0$
  - $y > 0$

Applying these Conditions to the given set of numbers, 28 faulty additions, 28 faulty substractions, 33 faulty multiplications and 41 faulty divisions are performed. In total $2^{3^2} * 8 = 512$ operations are performed, from which $2^{3^2} * 4 = 256$ are non primary. This results in an error rate of 25.4% for all operations or respectively an error rate of 50.8% for all non primary operations.

D. Change due to implemention of $0b111$ as "Undefined"

When implementing a specific combination of bits, as the undefined Token ("Undefined"), it has to be considered, what happens when executing operations on "Undefined". There are 2 obvious ways of dealing with this. The first one being, to consider every operation involving "Undefined" as the identity operation $x \mapsto x$. On the other hand, it can be defined that every operation involving "Undefined" results in undefined $(x, "Undefined") \mapsto "Undefined"$. For the following analysis, the second option is considerd.

Due to the implemention of "Undefined", the representable set of numbers in n bits has shrunken to $2^n - 1$, or in our case to $\{x \in \mathbb{Z} | 0 \leq x \leq 6\}$. However, if we define, that the result of a faulty operation, is "Undefined", the number of incorrect Operations, decreases by definition. While this is holds for amount of faulty operations, the amount amount of wrong numerical operations increases, as the Set of correct numerical calculations shrinks because, as amongst other things, the condition for correct results in addition shrinks to $x + y < 2^n - 1$ from $x + y < 2^n$.

E. Increasing the length of bits

When the numbersystem is increased to represent the set $\{x \in \mathbb{Z} | 0 \leq x < 2^4\}$, the number of faulty operations increases to 745, while the number of performed operations increases to $2^{4^2} * 8 = 2048$, which increases the percentage of faulty operations to 36.4%. Respectively for a 5 bit system, ther error percentage increases to 40.4%. This trend is very logical, as especially in multiplication, the number of invalid operations, increases considerably faster than the numbve rof valid operations. However the amount of valid operations in division increases with a bigger number System. This growth however is not strong enough to counteract the increase in invalid operations in multiplication.