Designing the First Many-valued Logic Computer

Ben Choi
Computer Science
Louisiana Tech University, USA
Email: pro@BenChoi.org

***Abstract*—This paper aims to design the first microprocessor based on many-valued logic circuits. The designed microprocessor will then form the basis for building the first many-valued logic computer. The aim is to design and build new computers entirely within the domain of many-valued logic. In a four-valued logic computer, each wire carries two bits at a time, each logic gate operates two bits at a time, and each memory cell records two bits at a time. This paper proposes a simple computational model by providing a simple architecture and by defining a simple instruction set. It then provides the design of the arithmetic and logic processing unit, a register file, a new many-valued memory cell, a new many-valued tri-state buffer, and a new decoder. In addition, it proposes a new methodology for designing any many-valued logic circuits. Having the design of the microprocessor and all the needed components provide the necessary and sufficient tools for exploiting the many-valued computational paradigm. We are now ready and are working to build the first many-valued microprocessor and computer.**

***Index Terms*—fuzzy computer, many-valued logic, fuzzy control, circuit design, fuzzy system**

I. Introduction

The performances of current computers are reaching their limits. Almost all present day computers are built based on two-valued logic. In two-valued logic, each wire can have two states. The performance of current computer depends mostly on how quickly the states can be changed, which determines the clock speed. During the past decades, the clock speed for CPU had doubled almost every year. In recent years, the clock speed doubled every 18 months. Now, it has become progressively difficult to increase the clock speed. The limit is approaching. Recently, CPU manufacturers are trying to circumvent the limitation of clock speed by packing more and more "cores" into a chip, which has resulted in dual-core or quad-core CPUs. However, this multi-core approach does not greatly improve the performance. This is due in part by the limit of the amount of data that can be transferred between the CPU and its connected components, which is determined by the number of pins on the CPU. Using two-value logic each pin on the CPU can have at most two states, and again the amount of data that can be transferred is determined by the clock speed. Thus, the multi-core approach does not circumvent the limitation.

Thus, there is a need for an innovative approach in order to push the speed limit of computing. Now is the time to depart from the two-valued logic to venture into many-valued logic and even into infinite-valued (Fuzzy) logic. Advancing from two-valued to four-valued logic provides an progressive approach [1]. Four symbols {0, 1, 2, 3} are needed to distinguish the four values, as shown in Table 1. The four values may represent anything, for example, the four bases {A, T, C, G} found in DNA, or the probability {0, 1/3, 2/3, 1}. These four values can be converted to binary numbers {00, 01, 10, 11}, or they can simply represent integers {0, 1, 2, 3}.

TABLE I. Representations of a Four-valued Variable

Symbol	DNA	Probability	Bits	Integer
0	A	0	00	0
1	T	1/3	01	1
2	C	2/3	10	2
3	G	1	11	3

To fully exploiting the many-valued computational paradigm, it is necessary to start from the ground up by designing components needed for constructing many-valued logic circuits. For example, each four-valued logic gates will operate two bits of data at a time, and each memory cell will record two bits at once. Now, each wire or CPU pin can have four states, which could double the amount of data that can be transferred between the CPU and its connected components without increasing the number of pins on the CPU.

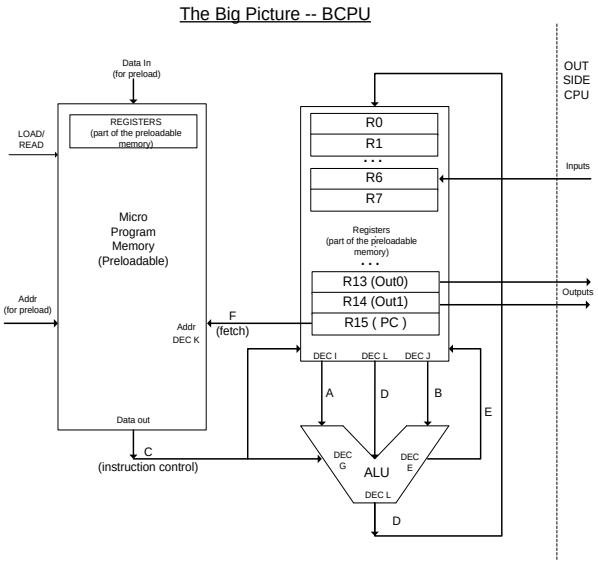

Figure 1. The High-level Architecture of the Many-valued Microprocessor

Manuscript received June 14, 2017; revised August 1, 2017.

doi: 10.18178/ijmerr.6.5.417-424

With eight-valued logic, each logic gate operates three bits of data and each CPU pin carries three bits of data at a time. The extreme case will be the infinite-valued or Fuzzy logic. Now, a different limit is being pushed.

The approach for using many-valued logic is in fact currently being employed in building higher capacity flash memory. The industry is pushing to allow each memory cell to store not just one bit, but two bits, three bits, and even four bits[2][3][4]. Now, we are proposing to push these limits not just in memory technologies, but also in computation.

TABLE II. INSTRUCTION FORMATS AND DEFINITIONS

Op-Code	Format	Definition
0000	MOVE R_D, R_A	$R_D \leftarrow R_A$
0001	NOT R_D, R_A	$R_D \leftarrow$ bitwise NOT R_A
0010	AND R_D, R_A, R_B	$R_D \leftarrow R_A$ bitwise AND R_B
0011	OR R_D, R_A, R_B	$R_D \leftarrow R_A$ bitwise OR R_B
0100	ADD R_D, R_A, R_B	$R_D \leftarrow R_A + R_B$
0101	SUB R_D, R_A, R_B	$R_D \leftarrow R_A - R_B$
0110	ADDI R_D, R_A, 4 bit data	$R_D \leftarrow R_A$ + 4 bit data
0111	SUBI R_D, R_A, 4 bit data	$R_D \leftarrow R_A - 4$ bit data
1000	SET R_D, 8 bit data	$R_D \leftarrow$ 8 0's follow by 8 bit data
1001	SETH R_D, 8 bit data	$R_D \leftarrow$ 8 bit data follow by R_{D7}, R_{D6} … R_{D0}
1010	INCIZ R_D, 4 bit data, R_B	$R_D \leftarrow R_D$ + 4 bit data, IF R_B == 0 (zero)
1011	DECIN R_D, 4 bit data, R_B	$R_D \leftarrow R_D - 4$ bit data, IF R_{B15} == 1 (neg)
1100	MOVEZ R_D, R_A, R_B	$R_D \leftarrow R_A$ IF R_B == 0 (zero)
1101	MOVEX R_D, R_A, R_B	$R_D \leftarrow R_A$ IF R_B != 0 (not zero)
1110	MOVEP R_D, R_A, R_B	$R_D \leftarrow R_A$ IF R_{B15} == 0 (positive)
1111	MOVEN R_D, R_A, R_B	$R_D \leftarrow R_A$ IF R_{B15} == 1 (negative)

To make the many-valued computation possible, this paper provides the necessary and sufficient tools and components for designing many-valued systems entirely within the domain of many-valued logic. The aim is to design the first microprocessor based on many-valued logic circuits and then to use the microprocessors to build many-valued logic computers. For serving as a prototype, the design will be kept simple. The author first proposes a simple computational model by providing a simple architecture and by defining a simple instruction set. Fig. 1 shows the high-level architecture of the many-valued microprocessor. At this high-level, the architecture looks similar to that of conventional digital computer. The use of many-valued logic is more explicit when the arithmetic and logic processing unit (ALU) is being designed and implemented. The author then proposes a design of a many-valued memory cells that are then used to design and implement the registers. The designs of many-valued tri-state buffers and decoders are also provided in this paper, which are then used to design and implement the control circuits. The design of the pre-loadable micro program memory is similar to that used for design multi-bit flash memory cells [2][3][4]. In addition, the author also proposes a methodology for designing any many-valued circuits. Equipped with the design details and methodology, we are now ready to venture into the domain of many-valued logic computation.

The remaining of this paper is organized as follows. Section II outlines the related research and their limitations. Section III provides the high-level design of the many-valued microprocessor, highlights the overall architecture, and defines the instruction set. Section IV proceeds on the design of the arithmetic and logic processing units. It also outlines a methodology for designing any many-valued circuits and provides the design of a four-valued adder circuit as an example. Section V describes the design of a many-valued memory cell that is used in the design of many-valued registers. It also provides the design of a four-valued tri-state buffer and decoder that are used for the design of the control units. Section VI discusses the implementation and programming aspects of the microprocessor. Section VII gives the conclusion and outlines the future research.

II. RELATED RESEARCH

To exploit the many-valued computation in hardware, we need the fundamental building blocks for many-valued logic circuits: many-valued logic gates, memory cells, and flip-flops. However, even these essential logic gates and memory cells are not yet fully developed. Currently, many-valued and fuzzy systems [5],[6],[7],[8],[9],[10],[11] are usually simulated or implemented by using a fuzzifier to convert the inputs into binary, using a set of fuzzy rulesfor processing and inferring, and using a defuzzifier to convert the binary results to outputs. To go a step further, researchers are now researching on many-valued and fuzzy logic circuits that can fully implement fuzzy systems.

To make the transition from two-valued to many-valued logic circuits, researchers were attempting to adapt CMOS [12],[13] technologies to implement the many-valued and Fuzzy logic gates. The design of the AND gate and the OR gate using CMOS technology was reported [1],[14],[15],[16]. Other technologies, including carbon nanotube and single electron transistors, have been attempted to build many-valued logic circuits

[17][18][19]. Other researchers used analog circuits to implement the many-valued and fuzzy logic gates [20],[21],[22]. However, these analog circuits were more difficult to be fabricated.

Many-valued and fuzzy memory cells or fuzzy flip-flops were proposed in [11],[23],[24],[25],[26],[27],[28],[29],[30],[31]. Concept of fuzzy flip-flop was first mentioned by Hirota [23]. They used analog gates [32],[33],[34]for the design their JK-type flip-flop as discussed in [19]. Hirota[23]defined fuzzy JK flip-flop based on the binary JK flip-flop but using fuzzy operators. Their design was based on fuzzy operators such as t-norm, s-norm, and fuzzy negation[35]. Virant et al.[29]proposed a design of T-type fuzzy flip-flop. The researchers adapted a strategy similar to Hirota [23] in the design of the T fuzzy flip-flop. However, we found that the fuzzy memory cells or flip-flops reported previously, such as JK-type flip-flop [23][24][25] and T-type flip-flop [29], have their limitations and cannot fully be used as general fuzzy memory cells. The flip-flops would not produce the correct results under certain input conditions[37].

In this paper, we focus on the design methodologies to utilize the proposed many-valued logic gates and memory cells to design many-valued logic computing systems[38][39], in particularly to design a many-valued microprocessor.

III. Designing the First Many-Valued Microprocessor

This section describes the top-level design of a simple many-valued microprocessor. At the top-level, the design process of a many-valued microprocessor is similar to that of a conventional digital microprocessor. The process begins by designing the computational model and defining the instruction set.

The high-level architecture of the microprocessor is shown in Fig. 1 (called as BCPU), which highlights the major components of the computational model. The processor consists of 16 registers (R0, R1, ... R15) for storing data. Within these 16 registers, there are 4 special purpose registers: R6 is the input register; R13 and R14 are the output registers; R15 is the program counter (PC). The processor consists of an ALU (arithmetic and logic unit) for preforming computations. It consists of a micro program memory for storing instructions. The micro program memory is a type of nonvolatile memory (like flash memory) that allows the instructions to be pre-loaded. The 16 registers is part of the address (0, 1, 2, … 15) space of the program memory, such allowing the contents of the registers to be re-loaded as well.

The process of preforming one computation specified by one instruction is outlined as follows. The program counter (PC) contains the address of the instruction to be executed. This address is passed to program memory through the F bus (as shown in Fig. 1). The program memory retrieves the instruction and outputs it on C bus. The wires of the C bus act as the control signals and pass to various decoders (DEC), that decode the instruction to select which registers and which processing unit (inside ALU) to be used to perform the instruction. The ALU receives the data of the selected registers through A, B, and D buses, performs the computation, and sends the result through D bus to be stored in the destination register. The ALU also sends enable signals through E bus to determine whether to write the result into the registers or not, which is used to implement conditional (if) statements. The final step is to update the contents of the program counter (PC), which will increase by one if it is not updated by the instruction. Any instructions writes into the program counter (PC) will function as Jump statement.

The microprocessor can perform 16 instructions that are defined as shown in Table 2. The first instruction is the MOVE operation that implements $R_D = R_A$, where R_D is one of the 16 registers (except the input register) for storing the result of the operation, and R_A is anyone of the 16 registers. For example, MOVE R2, R1 will results in R2 = R1 (contents of register R2 is replaced by contents of R1);SET R1, 100 will results in R1 = 100; and ADD R3, R2, R1 will results in R3 = R2 + R1.

One unique feature of this microprocessor is that anyone of these 16 instructions can write into the program counter PC (that is R15) and such can function as a Jump statement. For example, MOVE PC, R1 will jump to the address specified by R1, while MOVEZ PC, R1, R2 will jump only if R2 is zero (a conditional jump), and ADDI PC, PC, 8 will function as a relative jump for jumping forward.

Each instruction is specified by a 16-bit binary number. For example, ADD R3, R2, R1 is specified by a 4-bit op-code 0100 (as show in Table 2), R3 is coded by 0011, R2 by 0010, and R1 by 0001. Each of the 16 registers can store a 16-bit binary number. For serving as a prototype, the design is a simple 16-bit microprocessor.

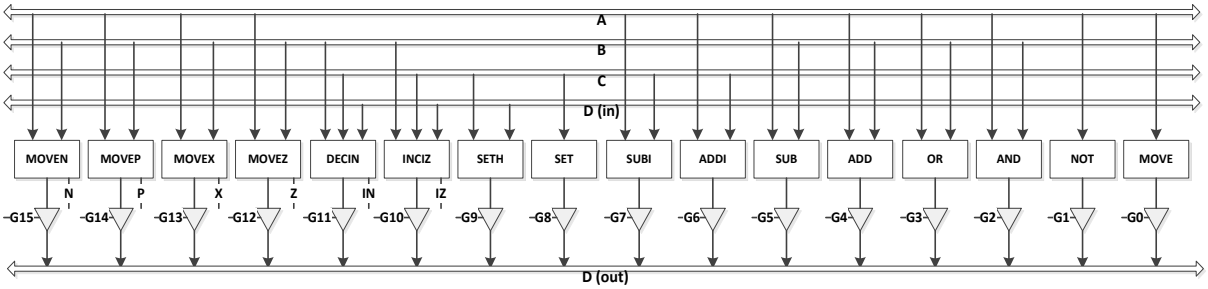

Figure 2. Arithmetic and Logic Processing Units (ALU)

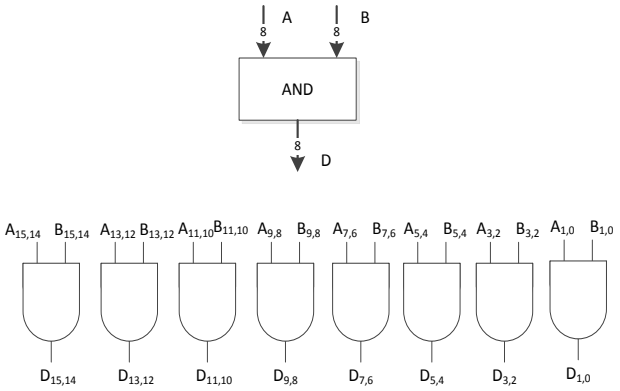

Figure 3. Implementation of a 16-bit AND Operation Using 4-valued Logic Circuits

The computational model and the instructions of this simple 16-bit microprocessor will serve as a prototype for design and implementation using many-valued logic circuits. One major advantage of using many-valued logic circuits to implement a microprocessor (or a computer) is to reduce the number of wires and components, as will be described in the remaining sections.

IV. Designing Many-valued Processing Units

This section begins to realize the design of the microprocessor by using many-valued logic circuits. It begins by implementing the processing units. The arithmetic and logic unit (ALU) of the processor consists of 16 processing units, each of which implements the operation of one instruction, as shown in Fig. 2. The processing units take input data from A, B, C, D buses, execute the specified operations, and pass the results to D output bus through tri-state buffers. There are 16 control signals (G0, G1, … G15) used to select which result should be available on the D output bus based on which instruction is being executed.

For this first design of a many-valued microprocessor, the implementation of the processing units will be done by using four-valued logic circuits, although sixteen-valued logic circuits would also be well suited. Each of the processing units will be implemented by using four-valued logic circuits: each wire can carry 2 bits of data at any state and each logic gate can operate 2 bits of data at a time. For example, the design of the AND processing unit is shown in Fig. 3. The AND operation takes 8 wires (realizing 16 bits) as input A and 8 wires as input B; performs the bit-wise AND operation using 8 four-valued AND gates; and outputs the results using 8 wires as D (realizing 16 bits outputs). The number of wires and gates reduces by 50%in comparing conventional digital microprocessors.

The OR and NOT processing units can be implemented using the same method as outlined for AND processing unit. The MOVE, SET, and SETH processing units only requires wires within for setting specific bits. The MOVEN, MOVEP, MOVEX, and MOVEZ also contains logic gates for checking the conditions.

The remaining processing units, ADD, SUB, ADDI, SUBI, INCIZ, and DECIN, all require the function of adding two numbers, where the SUB (A - B) is implemented as A + (-B).

A general design methodology will be outlined in the following, instead of providing the detailed design of the adding function. The methodology can be used to design any many-valued logic circuits, although the design of a four-valued adder is provided as an example.

The following outlines a simple four-step process for designing many-valued circuits to implement any many-valued functions[38]. The four steps are: (0) Creating a truth table to define the function; (1) Connecting each input x to n $C_i(x)$ gates; (2) Creating an AND gate for each output instance having a value > 0; and (3) Connecting the outputs of all the AND gates to an OR gate, which produces the outputs of the required function. These 4 steps are described in more details in the following:

Step 0.Truth Table: Creating a truth table to define the many-valued functions

As an example, we choose to design an adder that adds 2 four-valued numbers A, B. We create atruth tableto define the required functions,as shown in Table 3.All possible input combinations are shown in column A and B. The results of the addition is encoded by two outputs K and S, where K stands for carry and S stands for sum, and the total value is 4K+S. The column K defines the function required to produce K as output, and the column S defines the function required to produce S as output.

Step 1.$C_i(x)$ gates: Connecting each input x to n $C_i(x)$ gates for $0 \le i \le n-1$

TABLE III. Truth Table Defining A Four-valued Adder

Input		Output	
		4^1x	4^0x
A	B	K	S
0	0	0	0
0	1	0	1
0	2	0	2
0	3	0	3
1	0	0	1
1	1	0	2
1	2	0	3
1	3	1	0
2	0	0	2
2	1	0	3
2	2	1	0
2	3	1	1
3	0	0	3
3	1	1	0
3	2	1	1
3	3	1	2

Continuing the above example of designing an adder, the adder have two inputs, A and B. Now, we connect input A to 4 $C_i(A)$ gates:

$$C_0(A), C_1(A), C_2(A), C_3(A)$$

Similarly, we connect input B to 4 $C_i(B)$ gates:

$$C_0(B), C_1(B), C_2(B), C_3(B)$$

The results of these connection is shown in Fig. 4.

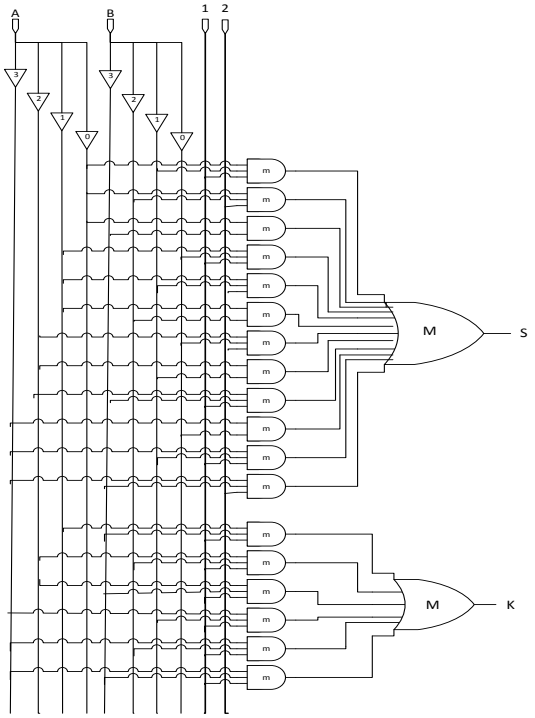

Figure 4. A Four-valued Adder Circuit for Building Arithmetic Operations

Step 2. AND gates: Creating an AND gate for each output instance having a value > 0

For eachinput instance$A_0,A_1,\ldots A_{m-1} = x_0,x_1,\ldots x_{m-1}$ that produce an output e> 0, create an AND gate connecting:

$$C_{x0}(A_0) \cdot C_{x1}(A_1) \cdot \ldots \cdot C_{xm-1}(A_{m-1}) \cdot e$$

For $e = e_{n-1}$, there is no need to connect the AND gate to e, which is the results of simplification based on the postulate P1 that is $e_{n-1} \cdot A = A$.

Continuing the example of designing an adder, for the function that produce S as output (in the S column of the truth table), there are 9 instances that produce output e > 0. For example, referring to the truth table, when inputs A=0, B=1, the output S=1, in this case we create an AND gate connecting: $C_0(A) \cdot C_1(B) \cdot 1$, in which since A=0 so the AND gate connects to the output of $C_0(A)$ gate (from Step 1), since B=1 so the AND gate connects to the output of $C_1(B)$ gate (from Step 1), and since S=1 so the AND gate connects to 1. When inputs A=0, B=2, the output S=2, in this case we create an AND gate connecting: $C_0(A) \cdot C_2(B) \cdot 2$,in which since A=0 so the AND gate connects to the output of $C_0(A)$ gate, since B=2 so the AND gate connects to the output of $C_2(B)$ gate, and since S=2 so the AND gate connects to 2. And, when inputs A=0, B=3, the output S=3, in this case we create an AND gate connecting: $C_0(A) \cdot C_3(B) \cdot 3$, which is simplified to $C_0(A) \cdot C_3(B)$. We create 9 AND gates for the 9 instances as shown in the below and the connections are shown in Fig. 4.

$C_0(A) \cdot C_1(B) \cdot 1$, $C_0(A) \cdot C_2(B) \cdot 2$, $C_0(A) \cdot C_3(B)$,
$C_1(A) \cdot C_0(B) \cdot 1$, $C_1(A) \cdot C_1(B) \cdot 2$, $C_1(A) \cdot C_2(B)$,
$C_2(A) \cdot C_0(B) \cdot 2$, $C_2(A) \cdot C_1(B)$, $C_2(A) \cdot C_3(B) \cdot 1$,
$C_3(A) \cdot C_0(B)$, $C_3(A) \cdot C_2(B) \cdot 1$, $C_3(A) \cdot C_3(B) \cdot 2$

Similarly, for the function that produce K as output (in the K column of the truth table), there are 6 instances that produce output e > 0. We create 6 AND gates as shown in the below and the connections are shown in Fig. 4.

$C_1(A) \cdot C_3(B) \cdot 1$, $C_2(A) \cdot C_2(B) \cdot 1$, $C_2(A) \cdot C_3(B) \cdot 1$,
$C_3(A) \cdot C_1(B) \cdot 1$, $C_3(A) \cdot C_2(B) \cdot 1$, $C_3(A) \cdot C_3(B) \cdot 1$

Step 3: OR gate:Connecting the outputs of all the AND gates to an OR gate, which produces the outputs of the required function.

Finishing the example of designing an adder,for the function that produce S as output (in the S column of the truth table), we connect the outputs of all the 9 AND gates (from Step 2) to an OR gate, as defined below:

$S = C_0(A) \cdot C_1(B) \cdot 1 + C_0(A) \cdot C_2(B) \cdot 2 + C_0(A) \cdot C_3(B) +$
$C_1(A) \cdot C_0(B) \cdot 1 + C_1(A) \cdot C_1(B) \cdot 2 + C_1(A) \cdot C_2(B) +$
$C_2(A) \cdot C_0(B) \cdot 2 + C_2(A) \cdot C_1(B) + C_2(A) \cdot C_3(B) \cdot 1 +$
$C_3(A) \cdot C_0(B) + C_3(A) \cdot C_2(B) \cdot 1 + C_3(A) \cdot C_3(B) \cdot 2$

Similarly, for the function that produceK as output (in the K column of the truth table), we connect the outputs of all the 6 AND gates (from Step 2) to an OR gate, as defined below:

$K = C_1(A) \cdot C_3(B) \cdot 1 + C_2(A) \cdot C_2(B) \cdot 1 + C_2(A) \cdot C_3(B) \cdot 1 +$
$C_3(A) \cdot C_1(B) \cdot 1 + C_3(A) \cdot C_2(B) \cdot 1 + C_3(A) \cdot C_3(B) \cdot 1$

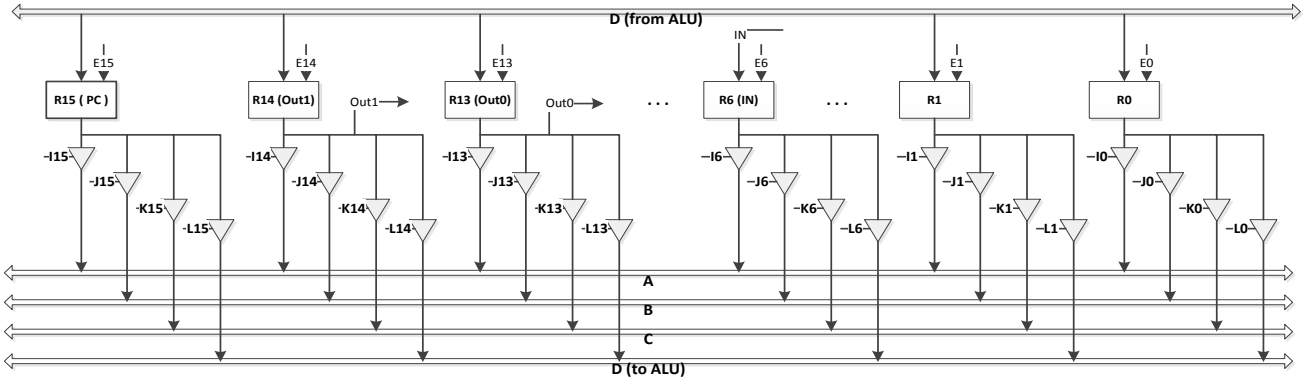

Figure 5. Register File and Data Buses

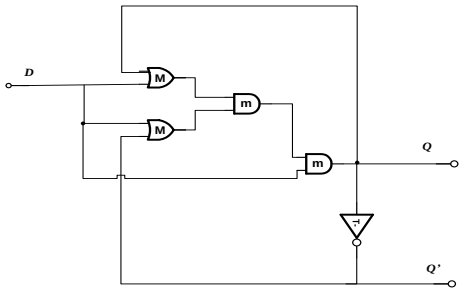

Figure 6. A Many-valued Memory Cell for Building Registers

The results all the connections are shown in Fig. 4, which is the four-valued circuit that implements the four-valued addition of two four-valued numbers.

To reduce the number of logic gates for a circuit, an additional step for minimization would be added into the steps outlined above. To further optimize the circuits for performance, specially designed transistor-level circuits would be used to implement the arithmetic processing units.

V. Designing Many-Valued Registers and Control Units

To continue the design of the many-valued microprocessor, this section describes the design of the registers and the control units. The high-level design of the register file is shown in Fig. 5. There are 16 registers, all of which (except R6) take inputs from ALU through the D bus. The input register R6 takes input from outside of the processor. Each register receives an enable signal that determine whether to write into the register, for implementing the conditional instructions. The registers store the results of the current step of computation and provide the data for later steps. Each register can provide its data on A, B, C, and D data buses. As shown in Fig. 5, the control signals, I, J, K, and L controlling the tri-state buffers, determine whether or not to provide the data on the buses. These data are available to the ALU for processing. Registers R13 and R14 also provide the data as outputs to the outside of the processor. Register R15 serves as the program counter (PC). Any instruction can write into the program counter to function as a Jump statement. If no instruction write into the PC, then it will increase by one after executing the current instruction.

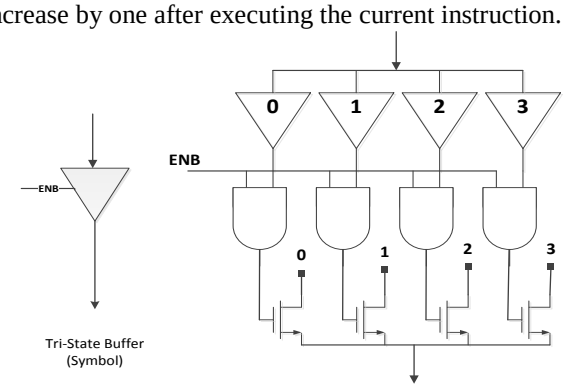

Figure 7. The Design of a Four-valued Tri-State Buffer

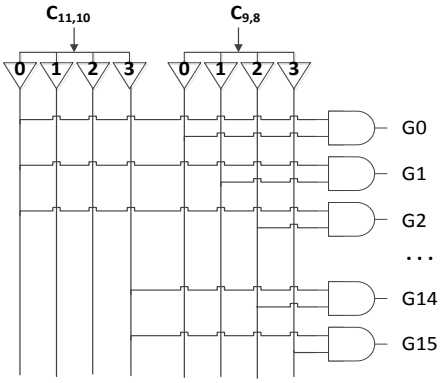

Figure 8. A Decoder from 2 Four-valued Wires to 16 Control Signals

To implement many-valued registers requires many-valued memory cells. The author has designed a memory cell that can store any many-valued data as shown in Fig. 6 [1]. The memory cell can be used for building the required registers. The memory cell is general purpose and the design used many-valued logic gates. To be specific in implementing the registers using four-valued logic, special transistor-level design would be possible for improving the performance.

Besides the many-valued memory cell, another key component needed is the many-valued tri-state buffer. The tri-state buffers are used in the register file (Fig. 5) for connecting the registers to the data buses, and are used in the ALU (Fig. 2) for connecting the processing units to the bus. The author provides a simple design concept of a four-valued tri-state buffer as shown in Fig. 7. If the enable signal (ENB) is False, there is no connection as all the 4 transistors (acting like switches) turn off. When ENB is True resulting in one of the switch turned on, the input value will be passed to the output.

The control units consist of many decoders, shown in Fig. 1 as DEC G, E, I, J, K, and L (located inside ALU, register file, and program memory). These decoders decode the instruction and produce many control signals. The design of the decoder DEC G is shown in Fig. 8. The decoder takes 2 four-valued wires and produces 16 control signals. The design of the remaining decoders are similar to this one.

VI. Implmentations and Programming

After completingthe design of the four-valued logic microprocessor, the next stage is the implementationfor building of the microprocessor and the computer. Before implementing the design in hardware, the author first created a simulator to test the processing and the instruction set. The simulator can execute assemble language programs written using the instruction set (defined in Table 2). An example program is shown in Fig. 9, that multiplies two numbers using a bitwise method. The testing results show that the instruction set is quite versatile despite the simplicity of the instructions and the overall architecture of the microprocessor.

```
// Multiply bitwise
// R3 = R1*R2

SET R3,0          // R3 = 0
SET R0,1          // Set test bit
AND R4, R0, R1    // Test bit n << Loop here
ADDI R5, PC,3     // Save jump addr
MOVEZ PC, R5, R4 // Skip next ADD if R4==0
ADD R3, R3, R2    // R3 = R3 + R2
ADD R0, R0, R0    // Shift left test bit
ADD R2, R2, R2    // Shift left R2
SUBI R5, PC, 6    // Save jump addr
MOVEP PC, R5, R0 // Loop back if R0 pos
```

Figure 9. A Program for Multiplication

To be able to realize the design of the microprocessor in hardware, all the needed many-valued logic components have been created and tested. These includes all the needed logic gates: AND, OR, NOT, and disjoint unary $C_i(x)$ gates; the needed memory cells, and the needed tri-state buffers. In additional, a methodology for designing any many-valued logic circuits have been developed. The design of adder circuits, the key component for implementing arithmetic operations, have been developed and tested. Now we are ready and are working to realize the design of many-valued microprocessor and the computer.

VII. Conclusion and Future Research

Now is the time to depart from the two-valued logic to venture into many-valued logic and even into infinite-valued or Fuzzy logic. To make many-valued computation possible, this paper provides the necessary tools for designing many-valued systems entirely within the domain of many-valued logic. It outlines the design of the first many-valued microprocessor, by providing design examples of the processing units, the registers, and the control units. To be able to implement these designs in hardware, the design of many-valued memory cell, tri-state buffer, and decoder are also provided. In addition, it describes a simple methodology for designing any many-valued circuits to implement any many-valued functions. Although not every pieces of details are provided in this paper, the overall design of the microprocessor, all the key components for implementation, and the design methodology should provide sufficient information for future developments. The next stage for future research will be to build the first many-valued microprocessor and computer.

References

[1] B. Choi, "Advancing from two to four valued logic circuits," *IEEE International Conference on Industrial Technology (ICIT 2013)*, February 2013.

[2] K. Kimura and T, Kobayashi, "Trends in high-density flash memory technologies," in *Proc. 2003 IEEE Conference on Electron Devices and Solid-State Circuits*, Dec. 2003, pp. 45-50.

[3] R. Jigour, "A tour of the basic of embedded NAND flash options," EE Times, Aug. 2013.

[4] Y. Choi, "NAND flash – The new era of 4 bit per cell and beyond," *EE Times*, May, 2009.

[5] P. Marinos, "Fuzzy logic and its application to switching systems," *IEEE Transactions on Computing*, vol. C-18, no. 4, p 343-348, Apr 1969.

[6] L. A. Zadeh, "Fuzzy Logic = Computing with words," *IEEE Transactions on Fuzzy Systems*, vol. 4, pp. 103-11, 1996.

[7] L. A. Zadeh, "The concept of linguistic variables and its application approximate reasoning," *Information Sciences*, pp. 43-80, 1975.

[8] J. M. Mendel, "Fuzzy logic systems for engineering: A tutorial," in *Proc. IEEE*, vol. 83, no. 3, March 1995.

[9] C. Isik, "Fuzzy logic: Principles, applications and perspectives," *SAE (Society of Automotive Engineers) Transactions*, vol. 100, n Sect 1 pt 1, 1991, 911148, pp. 393-396

[10] J. W. Fattaruso, S. S. Mahant Shetti, J. B. Barton, "A fuzzy logic inference processor," *IEEE Journal of Solid State Circuits*, vol. 29, no. 4, April 1994, pp. 397-402.

[11] G. Leslaw and J. Kluska, "Family of fuzzy J-K flip-flops based on bounded product, bounded sum and complementation," *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 28, no, 6, Dec. 1998, pp. 861-868.

[12] N. H. E. Weste and K. Eshraghian, *Principles of CMOS VLSI Design*, Addison-Wesley Publishing Company, 1994, pp. 61-69.

[13] Baker, R. Jacob, *CMOScircuitdesign, Layout, and Simulation*, 2nd *ed.* Baker, R. Jacob, 1964.

[14] V. Catania, A. Puliafito, M. Russo, and L. Vita, "A VLSI fuzzy inference processor based on a discrete analog approach," *IEEE Transactions on Fuzzy Systems*, vol. 2, no. 2, May 1994, pp. 93-106.

[15] G. Ascia, V. Catania, and M. Russo, M. "VLSI hardware architecture for complex fuzzy systems," *IEEE Transaction on Fuzzy Systems*, vol. 7, no. 5, Oct 1999, pp. 553-570.

[16] G. Ascia and V. Catania, "A high performance processor for application based on fuzzy logic," in *Proc. IEEE International on Fuzzy Systems Conference*, vol. 3, Aug. 22-25, 1999, pp. 1685-1690.

[17] A. Raychowdhury and K. Roy, "Carbon-Nanotube-Based Voltage-Mode Multiple-Valued Logic Design," *IEEE Trans. Nanotechnol*., vol. 4, no. 2, pp. 168–179, Mar. 2005.

[18] A. N. Sakhare and M. L. Keote, "Application of Galois field in VLSI using multi-valued logic," *Comput. Sci*., vol. 2, no. 1, 2013.

[19] G. Wu, L. Cai, and Q. Li, "Ternary logic circuit design based on single electron transistors," *Journal of Semiconductors*, vol. 30, no. 2, February 2009.

[20] K. Hirota, "Fuzzy logic and its Hardware implementation," in *Proc. 2nd New Zealand Two-stream international conference on Artificial Neural Networks and Expert systems (ANNES '95)*, annes, p. 102, 1995.

[21] J. L. Perez, M. A. Banuloes, "Electronic model on fuzzy gates," *Journal of the Mexican society of instrumentation*, vol. 3, NR 5, 1995, pp. 43-46.

[22] V. Catania and M. Russo, "Analog gates for a VLSI fuzzy processor" *8th International Conference of VLSI Design*, Jan 1995.

[23] K. Ozawa, K. Hirota, L. T. Koczy, W. Pedrycz, N. Ikoma, "Summary of fuzzy flip-flop," in International Joint Conference of the 4th IEEE International Conference on Fuzzy Systems and the 2nd International Fuzzy Engineering Symposium, vol. 3, 1995, pp. 1641-1648

[24] K. Hirota and K. Ozawa, "The concept of fuzzy flip-flop," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 19, no. 5, Sep-Oct, 1989, pp. 980-997.

[25] K. Hirota and W. Pedrycz, "Designing sequential systems with fuzzy J-K flip-flops," *Fuzzy Sets and Systems*, vol. 39, no. 3, Feb. 15, 1991, p. 261.

[26] D. McLeod, W. Pedrycz, J. Diamond, "Fuzzy JK flip-flops as computational structures: Design and implementation," *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 41, no. 3, Mar, 1994, pp. 215-226.

[27] K. Ozawa, K. Hirota, L. T. Koczy, K. Omori, "Algebraic fuzzy flip-flop circuits," *Fuzzy Sets and Systems*, vol. 39, no. 2, Jan. 25, p. 215, 1991.

[28] K. Hirota and W. Pedrycz, "Design of fuzzy systems with fuzzy flip-flops," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 25, no. 1, pp. 169-176, Jan. 1995.

[29] J. Virant, N. Zimic, and M. Mraz, "T-type fuzzy memory cells," *Fuzzy Sets and Systems*, vol. 102, no. 2, pp. 175-183, Mar 1, 1999,.
[30] S. M. Kia and S. Parmeswaran, "Designs for self checking flip-flops," *IEE Proceedings: Computers and Digital Techniques*, vol. 145, no, 2, Mar, 1998, pp. 81-88.
[31] T. Miki, T. Yamakawa, "Fuzzy inference on an analog fuzzy chip," *IEEE Micro*, pp. 8-18, 1995.
[32] T. Kettner, C, Heite, and K. Schumacher, "Analog CMOS realization of fuzzy logic membership functions," *IEEE Journal of Solid State Circuits*, vol. 28, no. 7, pp. 857-886, July 1993.
[33] H. Watanabe, W. D. Dettloff, and K. E. Yount, "A VLSI fuzzy logic controller with reconfigurable, cascadable architecture," *IEEE Journal Of Solid-State Circuits*., vol. 25, pp. 376-382, Apr. 1990.
[34] De Venuto, M. J. Ohletz, and B. Ricco, "Testing of analogue circuits via (standard) digital gates," in *Proc. International Symposium on Quality Electronic Design*, March 2002, pp. 112 – 119.
[35] I. Baturone, A. Barriga, S. Sanchez-Solano, and D. R. Lopez, "Microelectronic design of fuzzy logic-based systems," CRC Press, 2000.
[36] G. Epstein, *Multiple-Valued Logic Design: An Introduction*, Institute of Physics Publishing, 1993.
[37] B. Choi and K. Tipnis, "New components for building fuzzy logic circuits," *Fourth International Conference on Fuzzy Systems and Knowledge Discovery (FSKD 2007)*, vol. 2, pp. 586-590, 2007.
[38] B. Choi and K. Shukla, "Multi-valued logic circuit design and implementation," *International Journal of Electronics and Electrical Engineering*, vol. 3, no. 4, pp. 256-262, August 2015.
[39] B. Choi, R. Zheng, and K. Shukla, "Realizing many-valued logic for computation," *International Journal of Electronics and Electrical Engineering*, vol. 4, no. 4, pp. 227-283, August 2016.

Dr. Ben Choi has a Ph.D. degree in Electrical and Computer Engineering and also has a Pilot certificate for flying airplanes and helicopters. He is an Associate Professor in Computer Science at Louisiana Tech University. He received his Ph.D., M.S., and B.S. degrees from The Ohio State University, studied Computer Science, Computer Engineering, and Electrical Engineering. His areas of research include Humanoid Robots, Artificial Intelligence, Machine Learning, Intelligent Agents, Semantic Web, Data Mining, Fuzzy Systems, and Parallel Computing. His future research includes developing advanced software and hardware methods for building intelligent machines and theorizing the Universe as a Computer.