TABLE OF CONTENTS

PREPARATION

Section 1.1: Specifications

Firstly, it is important to know that a 4-bit unsigned adder will generate a maximum sum of $1111_2 + 1111_2 = 11110_2$, or in decimal, $15 + 15 = 30$.

Secondly, it is important to know what Logisim tools are accessible, and what their functions are. The basic logic gates, under the "Gates" category in Logisim, as well as multiplexers, demultiplexers, decoders, constants, and tri-state buffers are all accessible, with logic gates subject to a 5-bit fan-in limit. The adder, which can add two binary inputs together, and the comparator, which compares two binary inputs, are also accessible.

Two libraries are also available for use: a BCD-to-7-segment decoder library and the Logisim 7400 library. It is worth learning what each integrated circuit (IC) in the 7400 library does—while most are simply IC versions of already available logic gates with varying input and outputs, some may provide extra utility.

Table 1. Potentially useful 7400 ICs

IC number	**Purpose**
7448	BCD to 7-segment decoder
7483	4-bit binary full adder
74133	13-input NAND gate
74138	3-to-8 line decoder/demultiplexer
74371	256x8 ROM
74682	8-bit magnitude comparator

Note: upon completion of the project, only 7448 was used in implementation.

Components that are explicitly invalid are sequential and memory components such as flip-flops, ROMs, RAMs, and the 74371 IC.

Section 1.2: Addition

The project specifications allow the use of Logisim's built-in adder, which simplifies this section of the project.

The difficulty comes in displaying the 5-bit output in two 7-digit displays which, if done traditionally, requires a 5-to-14 decoder, or two 5-to-7 decoders. Such an implementation can be implemented through K-maps, as gate cost is not an issue in Logisim. Alternatively, the tens digit can be parsed through a series of comparators, leaving only the first digit to be decoded.

Converting any binary number to BCD to facilitate the implementation of a 7-segment decoder is not feasible as it would require an algorithm beyond the scope of the project called "double dabble".

Section 1.3: Divisibility rules

1. A number is divisible by 2 if the last digit is even.
2. A number is divisible by 3 if the sum of its digits is divisible by 3.
3. A number is divisible by 4 if its last two digits are divisible by 4.

In binary, a number is even if the rightmost bit is 0, and such, divisibility by 2 can be easily achieved.

The same cannot be said of the remaining two rules. For 3, every digit can be summed up, however checking divisibility creates a long recursion chain which only ends should the final sum be a number already known to be divisible by 3. For 4, the maximum number of digits of a 4-bit sum is already 2, so taking the last 2 digits of the number to be tested is no faster than performing a modulus operation.

Given the above considerations, the solution which comes to mind is a K-map for all possible numbers from 0 to 30 inclusive which are divisible by 3 and 4, of which there are only a total of 19 (11 numbers divisible by 3, and 8 numbers divisible by 4). For an input size of 5 bits (32 numbers), such a solution is still reasonable.

CALCULATIONS

Addition can be implemented through the adder component native to Logisim. The adder component outputs 4 bits as the truncated sum and a 5th bit for a carry-out. All that remains to be calculated are presenting the 5 bits as two digits and testing its divisibility. This 5-bit sum is represented as $ABCDE$ where A is MSB and E is the LSB.

Section 2.1: Tens digit

The tens digit can be easily implemented using comparators: 1 to compare between the sum and 10, another for 20, and another for 30.

Table 2. Values for the tens digit

Comparator 1 (≥ 10)	Comparator 2 (≥ 20)	Comparator 3 (≥ 30)	Tens digit
0	0	0	0
1	0	0	1
1	1	0	2
1	1	1	3

Table 2 can be implemented through constants and tri-state buffers:

Table 3. Truth table for tens digit tri-state buffers

Comparator 1 (≥ 10)	Comparator 2 (≥ 20)	Comparator 3 (≥ 30)	Buffer 1	Buffer 2	Buffer 3	Buffer 0
0	0	0	0	0	0	1
0	0	1	X	X	X	X
0	1	0	X	X	X	X
0	1	1	X	X	X	X
1	0	0	1	0	0	0
1	0	1	X	X	X	X
1	1	0	0	1	0	0
1	1	1	0	0	1	0

Table 4. Two-level forms for tens digit tri-state buffers

Component	**Two-level form (SOP)**
Buffer 1	$C1C2'$
Buffer 2	$C2C3'$
Buffer 3	$C3$
Buffer 0	$C1'$

Associated K-maps can be found under Section 5.1

A constant 1 is connected to buffer 1, a constant 2 is connected to buffer 2, a constant 3 is connected to buffer 3, and a constant 0 is connected to buffer 0. To convert the constants into a 7-segment format, either the 7448 IC or the BCD-to-7-segment decoder library can be used.

Using constants and tri-state buffers is simpler to implement than converting directly to 7-segment because the tens digit only needs to handle 4 digits (0 to 3) which require 4 K-maps as opposed to requiring 7 K-maps (1 for every segment). Additionally, this allows for the use of pre-made components to ease computation, as well as an opportunity to use components which have not been used before.

Section 2.2: Ones digit

There appears to be no easy way to parse the ones digit, so a full K-map is necessary for all 32 input combinations of the sum bits. While it is possible to also implement the ones digit using tri-state buffers, constants, and a binary-to-7-segment decoder, such an implementation will require 10 K-maps (one for every digit from 0 to 9) as opposed to 7 (one for every segment).

Table 5. Truth table for ones digit 7-segment display

A	B	C	D	E	a	b	c	d	e	f	g
0	0	0	0	0	1	1	1	1	1	1	0
0	0	0	0	1	0	1	1	0	0	0	0
0	0	0	1	0	1	1	0	1	1	0	1
0	0	0	1	1	1	1	1	1	0	0	1
0	0	1	0	0	0	1	1	0	0	1	1
0	0	1	0	1	1	0	1	1	0	1	1
0	0	1	1	0	1	0	1	1	1	1	1
0	0	1	1	1	1	1	1	0	0	0	0
0	1	0	0	0	1	1	1	1	1	1	1
0	1	0	0	1	1	1	1	1	0	1	1
0	1	0	1	0	1	1	1	1	1	1	0
0	1	0	1	1	0	1	1	0	0	0	0
0	1	1	0	0	1	1	0	1	1	0	1
0	1	1	0	1	1	1	1	1	0	0	1
0	1	1	1	0	0	1	1	0	0	1	1
0	1	1	1	1	1	0	1	1	0	1	1
1	0	0	0	0	1	0	1	1	1	1	1
1	0	0	0	1	1	1	1	0	0	0	0
1	0	0	1	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	1	0	1	1
1	0	1	0	0	1	1	1	1	1	1	0
1	0	1	0	1	0	1	1	0	0	0	0
1	0	1	1	0	1	1	0	1	1	0	1
1	0	1	1	1	1	1	1	1	0	0	1
1	1	0	0	0	0	1	1	0	0	1	1
1	1	0	0	1	1	0	1	1	0	1	1
1	1	0	1	0	1	0	1	1	1	1	1
1	1	0	1	1	1	1	1	0	0	0	0
1	1	1	0	0	1	1	1	1	1	1	1
1	1	1	0	1	1	1	1	1	0	1	1
1	1	1	1	0	1	1	1	1	1	1	0
1	1	1	1	1	0	1	1	0	0	0	0

Table 6. Two-level forms for ones digit 7-segment display

Segment	**Two-level form (POS)**
a	$(A+B+C'+D+E)(A'+B'+C+D+E)(A+B+C+D+E')(A'+B+C'+D+E')(A+B'+C+D'+E')(A'+B'+C'+D'+E')(A+B'+C'+D'+E)$
b	$(A'+B+C+D+E)(A+B+C'+D+E')(A'+B'+C+D+E')(A+B'+C'+D'+E')(A+B+C'+D'+E)(A'+B'+C+D'+E)$
c	$(A+B'+C'+D+E)(A+B+C+D'+E)(A'+B+C'+D'+E)$
d	$(A+B+C'+D+E)(A'+B'+C+D+E)(A+B+C'+D'+E')(A+B'+C'+D'+E)(A'+B+D+E')(A'+B'+D'+E')(B+C+D+E')(B'+C+D'+E')$
e	$(A+B+C'+D)(A+B'+C'+D')(A'+B'+C+D)(E')$
f	$(B+C+D+E')(B'+C+D'+E')(A'+B'+D'+E')(A+B+C+D')(A+B+D'+E')(A+B'+C'+D)(A'+B+C'+E')(A'+B+C'+D')$
g	$(B+C+D+E')(B'+C+D'+E')(A'+B'+D'+E')(A+B+C+D')(A+B+D'+E')(A+B'+C'+D)(A'+B+C'+E')(A'+B+C'+D')$

Associated K-maps can be found under Section 5.2

Some terms above can be written in terms of NAND gates:

$A'+B'+C'+D'+E'$	Basis
$(A'+B')+(C'+D')+E'$	Associativity
$(AB)'+(CD)'+E'$	De Morgan's
$((AB)'+(CD)')+E'$	Associativity
$(ABCD)'+E'$	De Morgan's
$(ABCDE)'$	De Morgan's

$$A'+B'+C'+D'+E' = (ABCDE)'$$
$$A'+B'+D'+E' = (ABDE)'$$
$$A'+B'+C'+D' = (ABCD)'$$

Section 2.3: Divisibility

As established in Section 1.3, divisibility by 2 is only dependent on the final bit. A truth table and K-map are only necessary for divisibility by 3 and 4.

Table 7. Truth table for divisibility by 3 and 4

A	B	C	D	E	By 3	By 4
0	0	0	0	0	1	1
0	0	0	0	1	0	0
0	0	0	1	0	0	0
0	0	0	1	1	1	0
0	0	1	0	0	0	1
0	0	1	0	1	0	0
0	0	1	1	0	1	0
0	0	1	1	1	0	0
0	1	0	0	0	0	1
0	1	0	0	1	1	0
0	1	0	1	0	0	0
0	1	0	1	1	0	0
0	1	1	0	0	1	1
0	1	1	0	1	0	0
0	1	1	1	0	0	0
0	1	1	1	1	1	0
1	0	0	0	0	0	1
1	0	0	0	1	0	0
1	0	0	1	0	1	0
1	0	0	1	1	0	0
1	0	1	0	0	0	1
1	0	1	0	1	1	0
1	0	1	1	0	0	0
1	0	1	1	1	0	0
1	1	0	0	0	1	1
1	1	0	0	1	0	0
1	1	0	1	0	0	0
1	1	0	1	1	1	0
1	1	1	0	0	0	1
1	1	1	0	1	0	0
1	1	1	1	0	1	0
1	1	1	1	1	0	0

Table 8. Table of two-level forms for divisibility by 3 and 4

Divisibility	**Two-level form (SOP)**
By 3	$A'B'C'D'E' + A'BCD'E' + ABC'D'E' + A'BC'D'E + AB'CD'E + A'B'C'DE + A'BCDE + ABC'DE + A'B'CDE' + AB'C'DE' + ABCDE'$
By 4	$D'E'$

Associated K-maps can be found under Section 5.3

Note that $D'E' = (D + E)'$ from De Morgan's law.

While divisibility rules for decimal numbers are difficult to implement in binary, it would appear that binary numbers have their own form of divisibility rules: in this case, it would appear divisibility by 4 is indicated by the 2 least significant bits. This can extend to divisibility by 8:

Table 9. K-map for divisibility by 8

		DE			
		00	01	11	10
ABC	000	1	0	0	0
	001	0	0	0	0
	011	0	0	0	0
	010	1	0	0	0
	100	1	0	0	0
	101	0	0	0	0
	111	0	0	0	0
	110	1	0	0	0

$SOP: C'D'E'$

From intuition, it seems that the n least significant bits of a binary number dictate its divisibility by 2^n. Proving or disproving such a statement is beyond the scope of the project, however.

Implementation

Section 3.1: Tens digit implementation

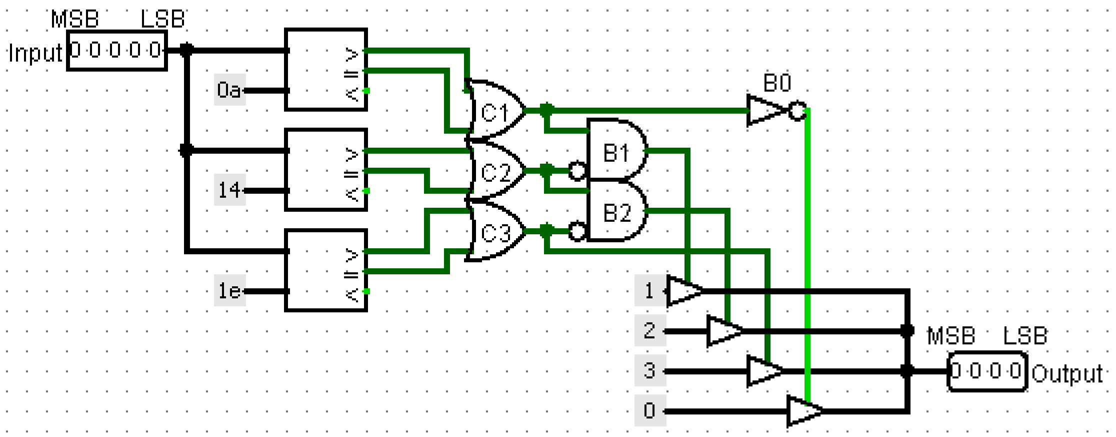

Figure 1. Implementation of the tens digit tri-state buffers two-level forms

A 5-bit input pin is attached to 3 comparators, which compare the input to 10, 20, and 30 (shown in hexadecimal as Logisim constants). 2-input OR gates are used to implement a greater-than-or-equal (≥) function to Logisim's comparators. The comparators are then used to implement the two-level forms found in Table 4. The output is now a 4-bit binary number equal to the tens digit in decimal.

Section 3.2: Ones digit implementation

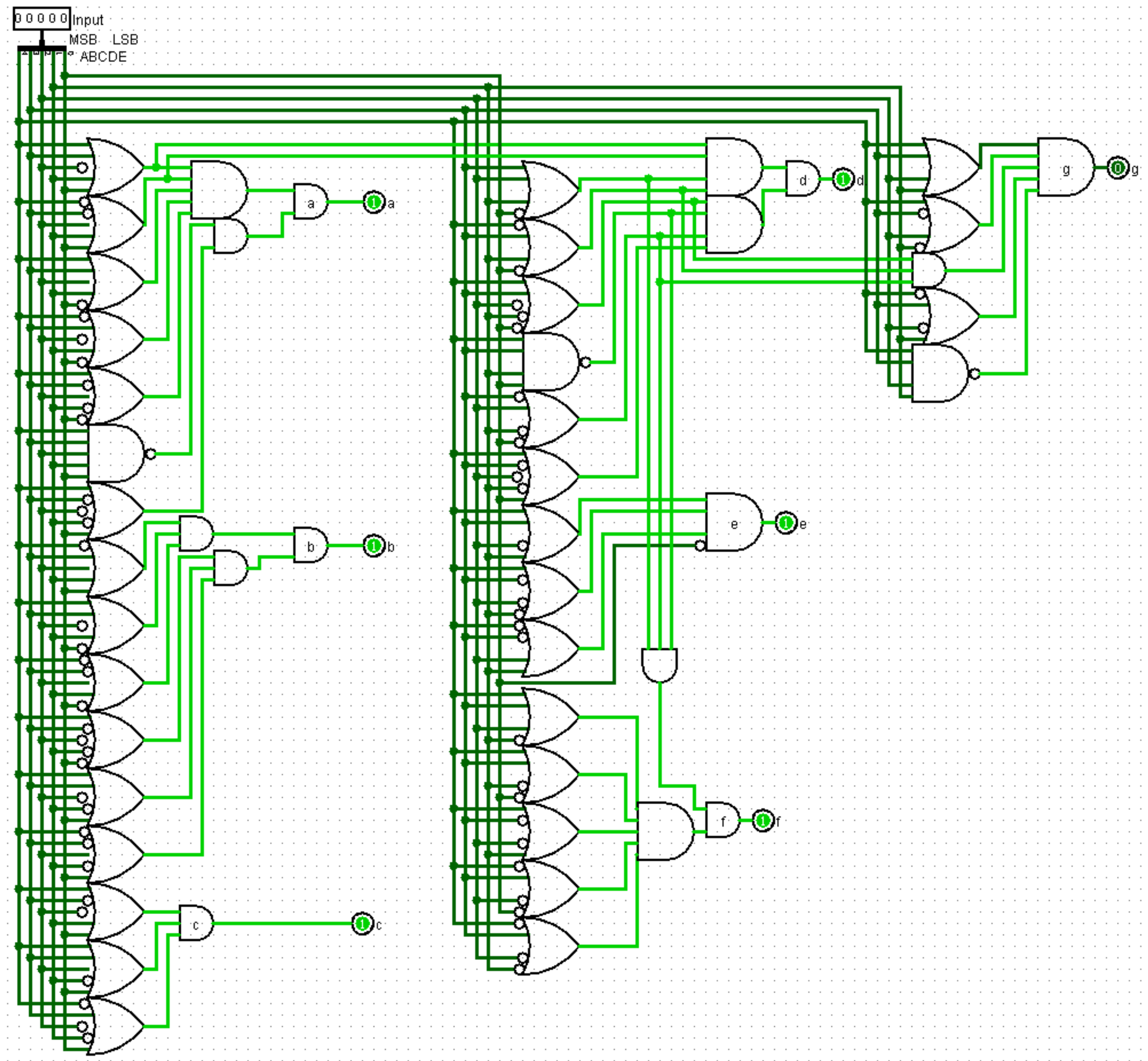

Figure 2. Implementation of the ones digit two-level forms

As is seen in Figure 2, the circuit for the ones digit is more complex than the circuit for the tens digit, largely because the ones digit requires 7 two-level forms. Some groupings are shared between segments, can be reused, as is the case for two groupings under a and four groupings under d.

Section 3.3: Divisibility implementation

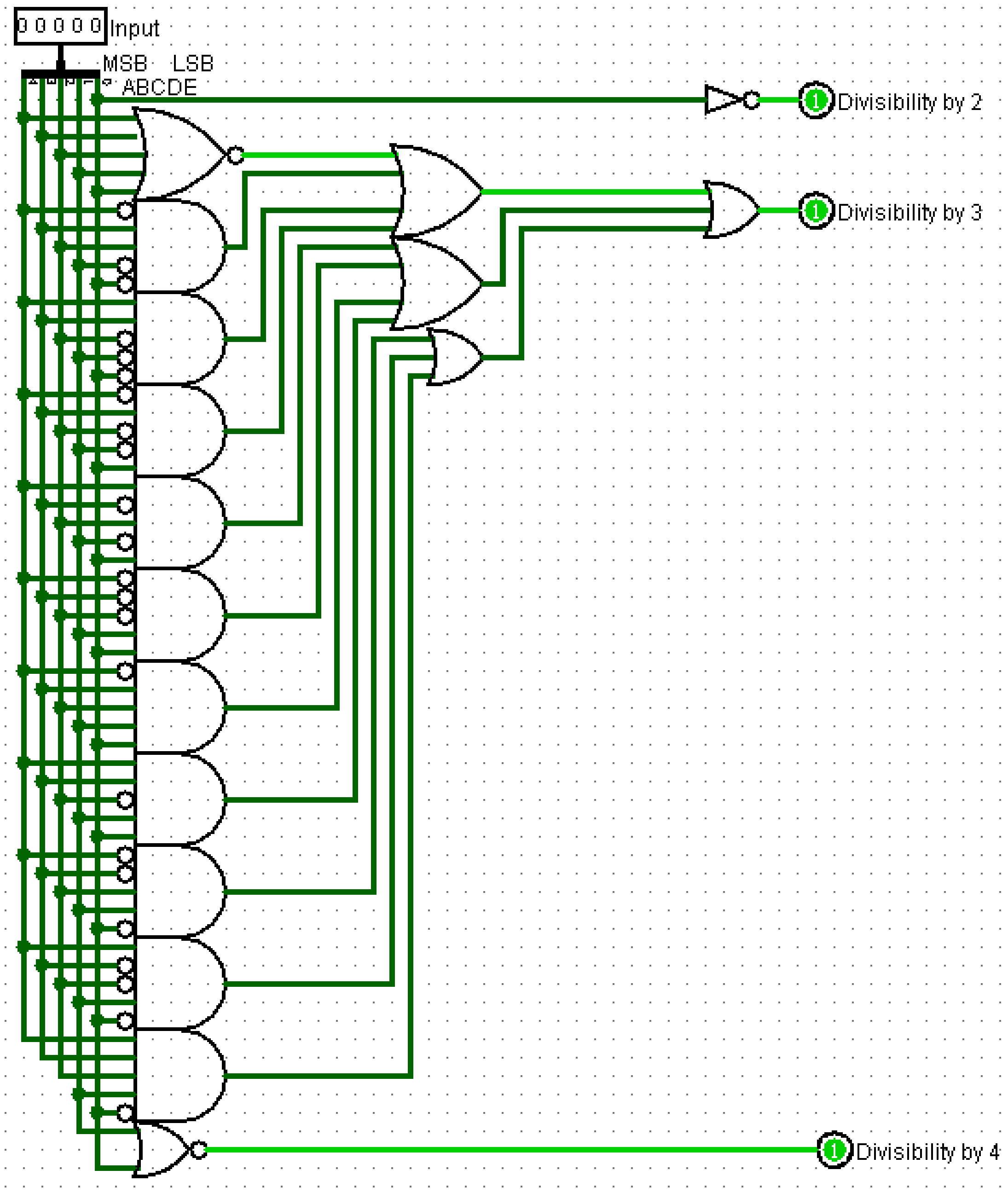

Figure 3. Implementation of divisibility two-level forms

As divisibility by 2 and 4 have simple two-level forms, they can both be implemented with only 1 gate each. Only divisibility by 3 has more than 1 level.

Section 3.4: High-level implementation

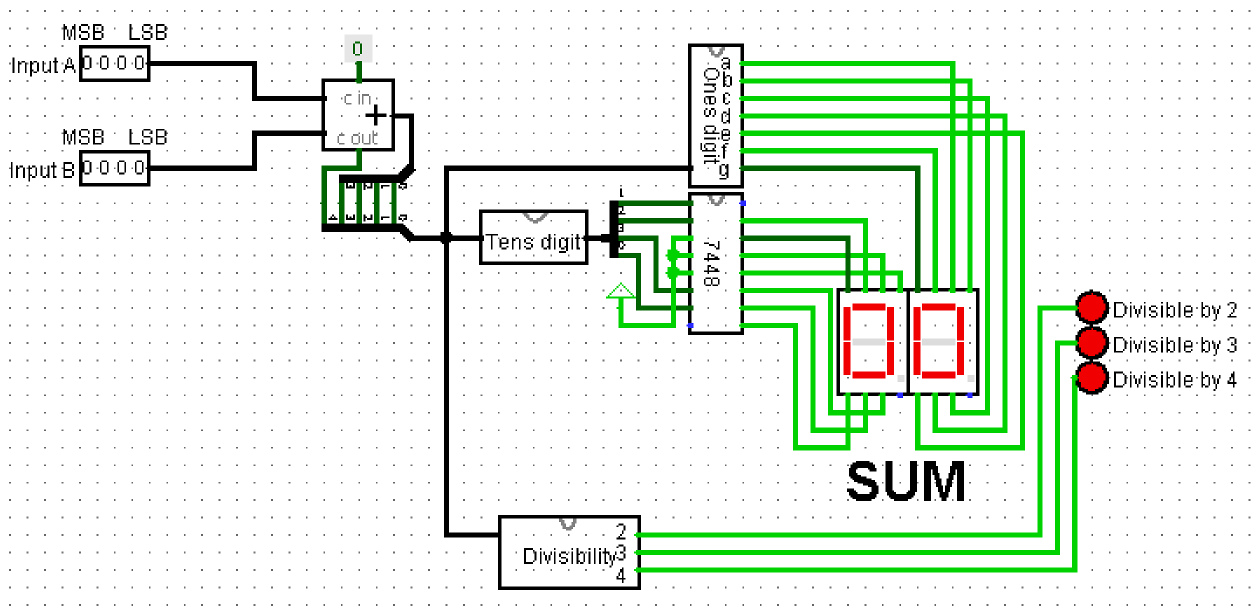

Figure 4. High-level implementation of the project

The two primary inputs, *A* and *B*, are connected to a Logisim adder. Their sum, denoted by the two outputs of the adder (truncated sum and carry-out), is then converted into one output through splitters. This output splits off to three ICs, which are circuits found in Figures 1, 2, and 3. This is possible through Logisim's built-in subcircuit function.

The ones digit, which decodes directly to every segment of a 7-segment display, is connected to a 7-segment display.

The tens digit outputs the digit of the tens digit, thus it must be connected to a separate decoder. This is achieved through a splitter and a 7448 IC. For the 7448 IC, two additional pins must be pulled HIGH for the decoder to function: Lamp-Test and Blanking Input, thus those pins are attached to a Logisim power source. A third pin, denoted as both Blanking Input and Ripple-Blanking Output, can also be pulled high, however the IC will function as intended even if the pin is left unconnected. The 7448 is then connected to a 7-segment display.

The divisibility circuit is attached to three LEDs which light up if the sum is divisible by two, three, or four.

REMARKS

A significant chunk of the project was completed before clarifications regarding valid components were announced, and as such, some areas of the project may be further simplified by using multiplexers or decoders. Additionally, it may be possible to further simplify the two-level forms through binary arithmetic, however this is not necessary as there are no limitations on gate number or circuit depth.

Video presentation can be found at https://drive.google.com/file/d/1xsTrNfQUyneCTxsR6PsXqi253GrxxhmV/view?usp=sharing

APPENDIX

Section 5.1: K-maps for tens digit calculations

Table 10. K-map for buffer 1

		C2C3			
		00	01	11	10
C1	0	0	X	X	X
	1	1	X	0	0

$SOP: C1C2'$

Table 11. K-map for buffer 2

		C2C3			
		00	01	11	10
C1	0	0	X	X	X
	1	0	X	0	1

$SOP: C2C3'$

Table 12. K-map for buffer 3

		C2C3			
		00	01	11	10
C1	0	0	X	X	X
	1	0	X	1	0

$SOP: C3$

Table 13. K-map for buffer 0

		C2C3			
		00	01	11	10
C1	0	1	X	X	X
	1	0	X	0	0

$SOP: C1'$

Section 5.2: K-maps for ones digit calculations

Table 14. K-map for a

		DE			
		00	01	11	10
ABC	000	1	0	1	1
	001	0	1	1	1
	011	1	1	1	0
	010	1	1	0	1
	100	1	1	1	1
	101	1	0	1	1
	111	1	1	0	1
	110	0	1	1	1

$$POS: (A + B + C' + D + E)(A' + B' + C + D + E)(A + B + C + D + E')(A' + B + C' + D + E')(A + B' + C + D' + E')(A' + B' + C' + D' + E')(A + B' + C' + D' + E)$$

Unannotated single 0-cells from top to bottom, left to right

Table 15. K-map for *b*

		DE			
		00	01	11	10
ABC	000	1	1	1	1
	001	1	0	1	0
	011	1	1	0	1
	010	1	1	1	1
	100	0	1	1	1
	101	1	1	1	1
	111	1	1	1	1
	110	1	0	1	0

$POS: (A' + B + C + D + E)(A + B + C' + D + E')(A' + B' + C + D + E')(A + B' + C' + D' + E')(A + B + C' + D' + E)(A' + B' + C + D' + E)$

Unannotated single 0-cells from top to bottom, left to right

Table 16. K-map for *c*

		DE			
		00	01	11	10
ABC	000	1	1	1	0
	001	1	1	1	1
	011	0	1	1	1
	010	1	1	1	1
	100	1	1	1	1
	101	1	1	1	0
	111	1	1	1	1
	110	1	1	1	1

$POS: (A + B' + C' + D + E)(A + B + C + D' + E)(A' + B + C' + D' + E)$

Unannotated single 0-cells from top to bottom, left to right

Table 17. K-map for *d*

		DE			
		00	01	11	10
ABC	000	1	0	1	1
	001	0	1	0	1
	011	1	1	1	0
	010	1	1	0	1
	100	1	0	1	1
	101	1	0	1	1
	111	1	1	0	1
	110	0	1	0	1

$POS: (A + B + C' + D + E)(A' + B' + C + D + E)(A + B + C' + D' + E')(A + B' + C' + D' + E)(A' + B + D + E')(A' + B' + D' + E')(B + C + D + E')(B' + C + D' + E')$

Blue box outline, green box outline, orange box outline, yellow box outline, blue box, green box, orange circle, blue circle

Table 18. K-map for *e*

		DE			
		00	01	11	10
ABC	000	1	0	0	1
	001	0	0	0	1
	011	1	0	0	0
	010	1	0	0	1
	100	1	0	0	1
	101	1	0	0	1
	111	1	0	0	1
	110	0	0	0	1

$POS: (A + B + C' + D)(A + B' + C' + D')(A' + B' + C + D)(E')$

Yellow box, orange box, green box, green circle

Table 19. K-map for *f*

		DE			
		00	01	11	10
ABC	000	1	0	0	0
	001	1	1	0	1
	011	0	0	1	1
	010	1	1	0	1
	100	1	0	1	1
	101	1	0	0	0
	111	1	1	0	1
	110	1	1	0	1

$POS: (B + C + D + E')(B' + C + D' + E')(A' + B' + D' + E')(A + B + C + D')(A + B + D' + E')(A + B' + C' + D)(A' + B + C' + E')(A' + B + C' + D')$

Orange circle, blue circle, green box, orange box, blue box, gray box, yellow box, red box

Table 20. K-map for *g*

		DE			
		00	01	11	10
ABC	000	0	0	1	1
	001	1	1	0	1
	011	1	1	1	1
	010	1	1	0	0
	100	1	0	1	1
	101	0	0	1	1
	111	1	1	0	0
	110	1	1	0	1

$POS: (A + B + C + D)(A + B' + C + D')(A + B + C' + D' + E')(A' + B + D + E')(B' + C + D' + E')(A' + B + C' + D)(A' + B' + C' + D')$

Green box, orange box, orange box outline, blue box, blue circle, red box, yellow box

Section 5.3: K-maps for divisibility calculations

Table 21. K-map for divisibility by 3

		DE			
		00	01	11	10
ABC	000	1	0	1	0
	001	0	0	0	1
	011	1	0	1	0
	010	0	1	0	0
	100	0	0	0	1
	101	0	1	0	0
	111	0	0	0	1
	110	1	0	1	0

$$SOP: A'B'C'D'E' + A'BCD'E' + ABC'D'E' + A'BC'D'E + AB'CD'E + A'B'C'DE + A'BCDE$$
$$+ ABC'DE + A'B'CDE' + AB'C'DE' + ABCDE'$$

Unannotated single 1-cells from top to bottom, left to right

Table 22. K-map for divisibility by 4

		DE			
		00	01	11	10
ABC	000	1	0	0	0
	001	1	0	0	0
	011	1	0	0	0
	010	1	0	0	0
	100	1	0	0	0
	101	1	0	0	0
	111	1	0	0	0
	110	1	0	0	0

$POS: D'E'$