CS20 Project 1:
BMI Calculator

Submitted by:
Batistil, Mansur Khalil (202315263),
Mislang, Ethan Renell Sebastian (202301491),
Mislang, Gabriel (202305587),
Peralta, Juan Marco (202315337)

1 LogiSim Component

1.1 Specifications and BMI Values

Before implementing the circuit for the LogiSim component, the group firstly organized the BMI values with respect to the ranges of Height (152 cm to 215 cm inclusive) and Weight (34 kg to 97 kg inclusive) accordingly in a table (see Figure 1.1.1).

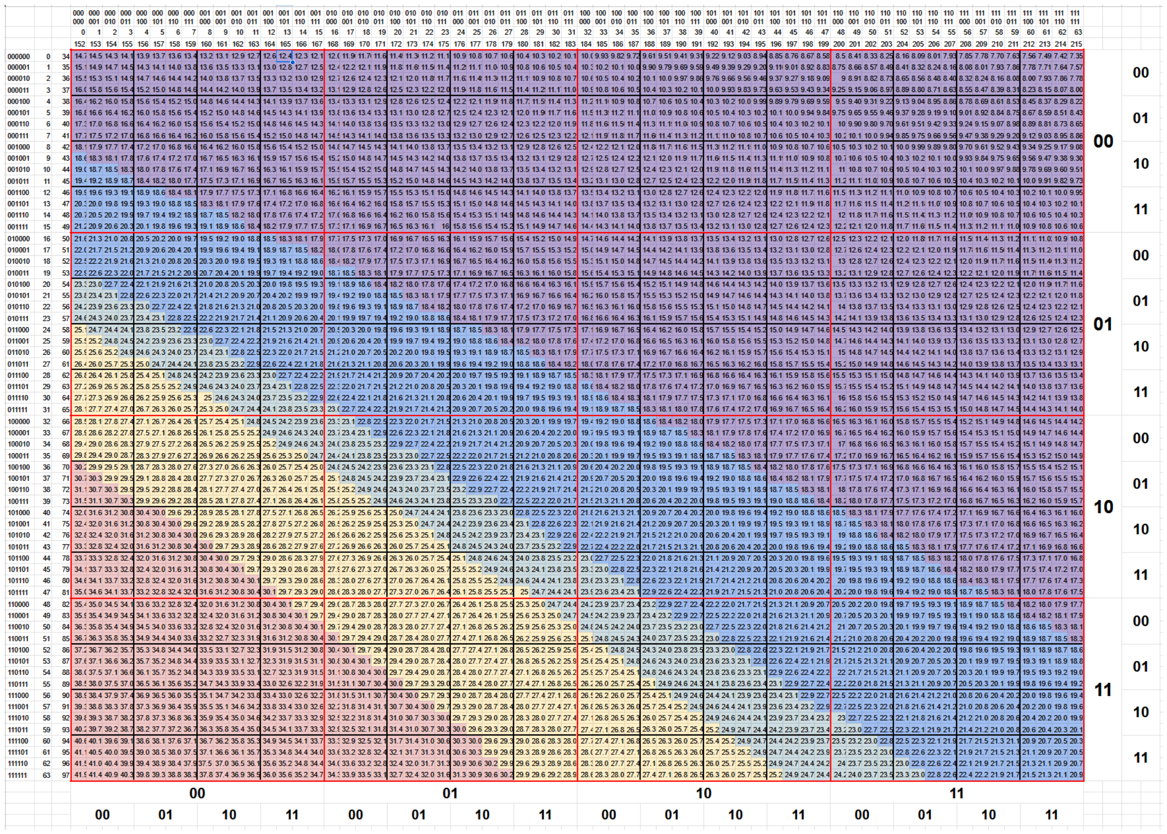


Figure 1.1.1. Table of BMI values with respect to the Height and Weight ranges

Figure 1.1.1 shows all 4096 (*64 x 64*) BMI calculated values for the fixed ranges of ***Weight*** and ***Height,*** with ***Weight*** as *rows* and ***Height*** as *columns* of the table.

Additionally, the group was able to classify them by values, highlighting and labelling them as follows:

- Purple - Underweight (*x1*)
- Blue - Normal (*x2*)
- Green - Overweight (*x3*)
- Yellow - Obese Class I (*x4*)
- Red - Obese Class II (*x5*)

As per the CS 20 Project 1 specification, the ***Weight*** and ***Height*** values should be encoded with *6 bits* each, with the ranges 152-215 and 34-97 mapped to the bit patterns 000000, 000001, 000010, and so on up to 111111. For future reference, the *6 bits* of ***Weight*** and ***Height*** in our system will be labelled as follows:

- ***Weight***: *W5W4W3W2W1W0*, with *W5* being the MSB, and *W0* being the LSB
- ***Height***: *H5H4H3H2H1H0*, with *H5* being the MSB, and *H0* being the LSB

With that being said, Figure 1.1.1 also showed the "*indexing*" of the table. By indexing, we mean the ***big*** and ***small*** indexing found at the right and bottom side of the table. The right side indexing refers to the "indexing" of ***Weight***. Its indexing is composed of two things, one for the ***big indexing*** and one for the ***small indexing***. The same goes for the bottom side "indexing," but this time, it refers to the "indexing" of ***Height***.

- ***big indexing*** – the four values (00, 01, 10, 11) which are the *first* two most significant bits of the ***Weight***, i.e. *W5W4* (or for the ***Height***, i.e. *H5H4*).
- ***small indexing*** – the four values (00, 01, 10, 11) under ***big indexing*** which are the *second* two most significant bits of the ***Weight***, i.e. *W3W2* (or for the ***Height***, i.e. *H3H2*).

The reason why it's called ***big indexing*** and ***small indexing*** is because of their behavior to group the KMaps into *subcircuits/subsystems*. These so-called subsystems are also composed of two types:

- ***big subsystem*** – the bigger subsystem (i.e. the 16 bigger subsystem divided by the red lines in Figure 1.1.1) only indexed by the ***big indexing***
- ***small subsystem*** – the smaller subsystem (i.e. the 16 smaller 4 by 4 subsystems divided by the black lines, and which are inside the bigger subsystems) only indexed by the ***small indexing***

Note: These two types would have their own SOP in determining which 4 by 4 KMap to "activate." So in a way, you first need to index the ***big subsystem*** by ***big indexing***. Then, it has to pass through the ***small subsystem*** by ***small indexing***. Then once you access the ***small subsystem***, it contains a regular 4 by 4 KMap.

1.2 Approach

Since the table in Figure 1.1.1 would have several ***big subsystems*** and ***small subsystems*** which can be indexed by the significant bits of ***Weight*** and ***Height***, we like to imagine our approach the same way you would think of *2D array indexing***.**

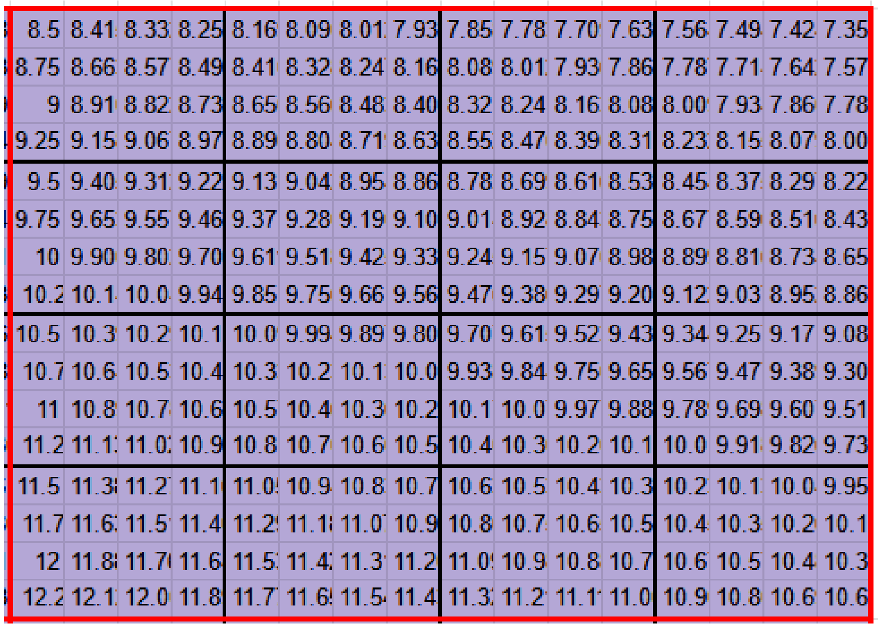
Let's say in Python, you initialized an m by n grid (i.e. *grid*[m][n]), wherein m are the rows, and n are the columns. For you to "access" each cell you would need to call the grid together with the cell's corresponding index (e.g. *grid*[i][j] for some integer $i \in [0, m)$ and for some integer $j \in [0, n)$.

The same analogy would go for our circuit. The *grid* would refer to the table in Figure 1.1.1, index i ∈ [0000, 1111 ; such that the 4 bits are formed as *W5W4H5H4*] would refer to the ***big indexing***, index j ∈ [0000, 1111 ; such that the 4 bits are formed as *W3W2H3H2*] would refer to the ***small indexing***, and *grid*[i][j] would refer to a regular 4 by 4 KMap type.

1.3 Example "Call" Function

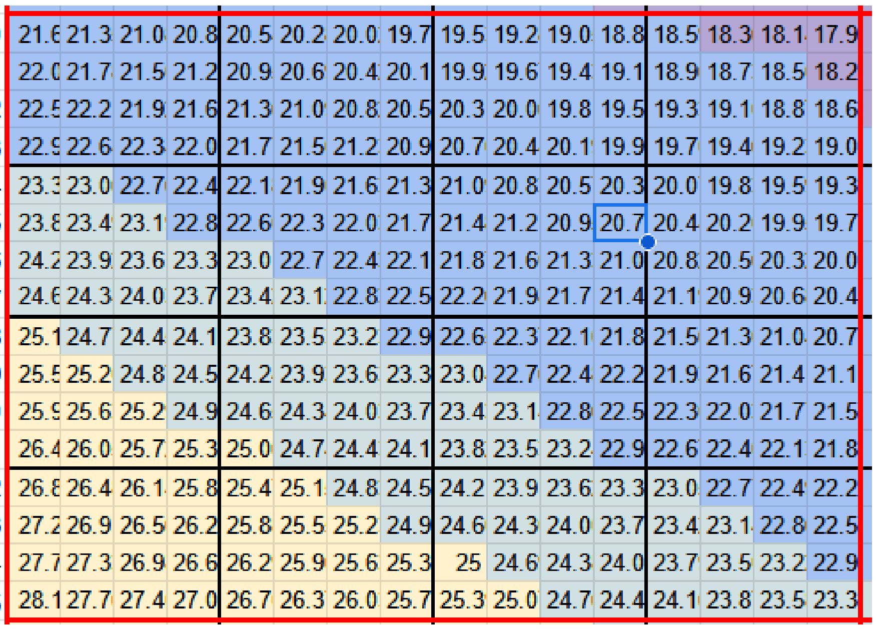
For example, if we were to access the *big_subsystem_0011* (i.e. the ***big subsystem*** at Figure 1.3.1), you would have to "call" *grid*[*0011*] only. Specifically, *W5W4H5H4* are *0011* respectively. However, if you want to access a regular 4 by 4 KMap type inside the *big_subsystem_0011*, you would need to call *grid*[*0011*][j]. The index j would be read as from left to right, then from top to bottom. So, if you were to access second to the right at the bottommost 4 by 4 KMap in the *big_subsystem_0011*, you would have to "call" grid[*0011*][*1110*]. Specifically, *W3W2H3H2* are *1110* respectively. Then, for the 4 by 4 KMap SOP by itself, we don't have to implement it since *x1* is just 1 (or turned on) in all cases.

Another example is if we were to access *big_subsystem_0100* (i.e. the ***big subsystem*** at Figure 1.3.2), you would have to "call" *grid*[*0100*]. Specifically, *W5W4H5H4* are *0100* respectively. However, if you want to access a regular 4 by 4 KMap type inside the *big_subsystem_0100*, you would need to call *grid*[*0100*][j]. Unlike the example earlier, this ***big subsystem*** contains different KMap types (see 1.6 Underlying Patterns and KMap types for more). With this, you need to implement different SOPs for each ***small indexing*** of a corresponding KMap type (see 1.5 Small Indexing before proceeding). Now, for you to access the *kmap_type_1* with yellow as ***on_color*** and green as ***off_color***, you have to "call" *grid*[*0100*][*1000*]. The reason for having KMap types is that you can also reuse *kmap_type_1* with green as ***on_color*** and blue as ***off_color*** in *grid*[*0100*][*1111*]. Note that for each *grid*[i][j] call refers to a KMap SOP expression and a circuit implementation based on ***big indexing*** and ***small indexing***.



8.5	8.41	8.33	8.25	8.16	8.09	8.01	7.93	7.85	7.78	7.70	7.63	7.56	7.49	7.42	7.35
8.75	8.66	8.57	8.49	8.41	8.32	8.24	8.16	8.08	8.01	7.93	7.86	7.78	7.71	7.64	7.57
9	8.91	8.82	8.73	8.65	8.56	8.48	8.40	8.32	8.24	8.16	8.08	8.00	7.93	7.86	7.78
9.25	9.15	9.06	8.97	8.89	8.80	8.71	8.63	8.55	8.47	8.39	8.31	8.23	8.15	8.07	8.00
9.5	9.40	9.31	9.22	9.13	9.04	8.95	8.86	8.78	8.69	8.61	8.53	8.45	8.37	8.29	8.22
9.75	9.65	9.55	9.46	9.37	9.28	9.19	9.10	9.01	8.92	8.84	8.75	8.67	8.59	8.51	8.43
10	9.90	9.80	9.70	9.61	9.51	9.42	9.33	9.24	9.15	9.07	8.98	8.89	8.81	8.73	8.65
10.2	10.1	10.0	9.94	9.85	9.75	9.66	9.56	9.47	9.38	9.29	9.20	9.12	9.03	8.95	8.86
10.5	10.3	10.2	10.1	10.0	9.99	9.89	9.80	9.70	9.61	9.52	9.43	9.34	9.25	9.17	9.08
10.7	10.6	10.5	10.4	10.3	10.2	10.1	10.0	9.93	9.84	9.75	9.65	9.56	9.47	9.38	9.30
11	10.8	10.7	10.6	10.5	10.4	10.3	10.2	10.1	10.0	9.97	9.88	9.78	9.69	9.60	9.51
11.2	11.1	11.0	10.9	10.8	10.7	10.6	10.5	10.4	10.3	10.2	10.1	10.0	9.91	9.82	9.73
11.5	11.3	11.2	11.1	11.0	10.9	10.8	10.7	10.6	10.5	10.4	10.3	10.2	10.1	10.0	9.95
11.7	11.6	11.5	11.4	11.2	11.1	11.0	10.9	10.8	10.7	10.6	10.5	10.4	10.3	10.2	10.1
12	11.8	11.7	11.6	11.5	11.4	11.3	11.2	11.0	10.9	10.8	10.7	10.6	10.5	10.4	10.3
12.2	12.1	12.0	11.8	11.7	11.6	11.5	11.4	11.3	11.2	11.1	11.0	10.9	10.8	10.6	10.6

Figure 1.3.1. *big_subsystem_0011*



21.6	21.3	21.0	20.8	20.5	20.2	20.0	19.7	19.5	19.2	19.0	18.8	18.5	18.3	18.1	17.9
22.0	21.7	21.5	21.2	20.9	20.6	20.4	20.1	19.9	19.6	19.4	19.1	18.9	18.7	18.5	18.2
22.5	22.2	21.9	21.6	21.3	21.0	20.8	20.5	20.3	20.0	19.8	19.5	19.3	19.1	18.8	18.6
22.9	22.6	22.3	22.0	21.7	21.5	21.2	20.9	20.7	20.4	20.1	19.9	19.7	19.4	19.2	19.0
23.3	23.0	22.7	22.4	22.1	21.9	21.6	21.3	21.0	20.8	20.5	20.3	20.0	19.8	19.5	19.3
23.8	23.4	23.1	22.8	22.6	22.3	22.0	21.7	21.4	21.2	20.9	20.7	20.4	20.2	19.9	19.7
24.2	23.9	23.6	23.3	23.0	22.7	22.4	22.1	21.8	21.6	21.3	21.0	20.8	20.5	20.3	20.0
24.6	24.3	24.0	23.7	23.4	23.1	22.8	22.5	22.2	21.9	21.7	21.4	21.1	20.9	20.6	20.4
25.1	24.7	24.4	24.1	23.8	23.5	23.2	22.9	22.6	22.3	22.1	21.8	21.5	21.3	21.0	20.7
25.5	25.2	24.8	24.5	24.2	23.9	23.6	23.3	23.0	22.7	22.4	22.2	21.9	21.6	21.4	21.1
25.9	25.6	25.2	24.9	24.6	24.3	24.0	23.7	23.4	23.1	22.8	22.5	22.3	22.0	21.7	21.5
26.4	26.0	25.7	25.3	25.0	24.7	24.4	24.1	23.8	23.5	23.2	22.9	22.6	22.4	22.1	21.8
26.8	26.4	26.1	25.8	25.4	25.1	24.8	24.5	24.2	23.9	23.6	23.3	23.0	22.7	22.4	22.2
27.2	26.9	26.5	26.2	25.8	25.5	25.2	24.9	24.6	24.3	24.0	23.7	23.4	23.1	22.8	22.5
27.7	27.3	26.9	26.6	26.2	25.9	25.6	25.3	25	24.6	24.3	24.0	23.7	23.5	23.2	22.9
28.1	27.7	27.4	27.0	26.7	26.3	26.0	25.7	25.3	25.0	24.7	24.4	24.1	23.8	23.5	23.3

Figure 1.3.2. *big_subsystem_0100*

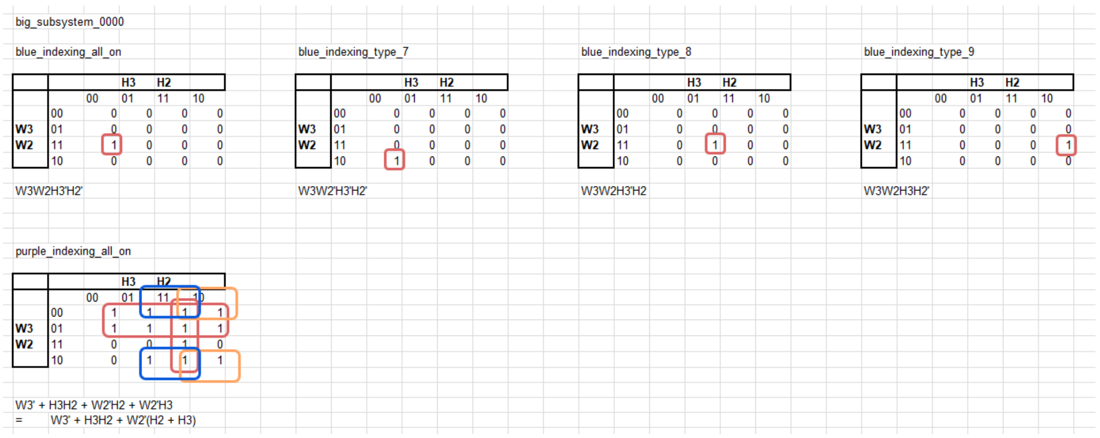
1.4 Big Indexing

The KMaps for ***big indexing*** is not anymore needed as it is *common sense* by their *codename*. For example, the ***big indexing*** SOP for *big_subsystem_0011* is just *W5'W4'H5H4*. Another example, the ***big indexing*** SOP for *big_subsystem_0100* is just *W5'W4H5'H4'*. So, if we were to KMap it, it's just only one '1' in the KMap table.

1.5 Small Indexing

Each KMap and SOP done for ***small indexing*** can be found in the *cs20project1.csv* file. However, due to how the CSV file operates, some formatting will be omitted. To make up for that case, we will include a more visually appealing KMaps and SOP for each ***big subsystem***'s ***small indexing*** for each KMap type (see Figure 1.5.1). Note that there's no need to implement KMaps for *big_subsystem_0001*, *big_subsystem_0010*, *big_subsystem_0011*, and *big_subsystem_0111* since all are all on for *x1*.

Each of these SOPs are then implemented using color_indexing_template (see Figure 1.5.2).



big_subsystem_0000

blue_indexing_all_on

W3 W2	H3 H2: 00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	1	0	0	0
10	0	0	0	0

W3W2H3'H2'

blue_indexing_type_7

W3 W2	H3 H2: 00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	0	0	0	0
10	1	0	0	0

W3W2'H3'H2'

blue_indexing_type_8

W3 W2	H3 H2: 00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	0	1	0	0
10	0	0	0	0

W3W2H3'H2

blue_indexing_type_9

W3 W2	H3 H2: 00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	0	0	0	1
10	0	0	0	0

W3W2H3H2'

purple_indexing_all_on

W3 W2	H3 H2: 00	01	11	10
00	1	1	1	1
01	1	1	1	1
11	0	0	1	0
10	0	1	1	1

W3' + H3H2 + W2'H2 + W2'H3
= W3' + H3H2 + W2'(H2 + H3)

big_subsystem_0100

yellow_indexing_all_on

			H3	H2	
		00	01	11	10
	00	0	0	0	0
W3	01	0	0	0	0
W2	11	1	0	0	0
	10	0	0	0	0

W3W2H3'H2'

yellow_indexing_type_1

			H3	H2	
		00	01	11	10
	00	0	0	0	0
W3	01	0	0	0	0
W2	11	0	0	0	0
	10	1	0	0	0

W3W2'H3'H2'

yellow_indexing_type_2

			H3	H2	
		00	01	11	10
	00	0	0	0	0
W3	01	0	0	0	0
W2	11	0	1	0	0
	10	0	0	0	0

W3W2H3'H2

yellow_indexing_sp_type_3

			H3	H2	
		00	01	11	10
	00	0	0	0	0
W3	01	0	0	0	0
W2	11	0	0	0	0
	10	0	1	0	0

W3W2'H3'H2

yellow_indexing_type_11

			H3	H2	
		00	01	11	10
	00	0	0	0	0
W3	01	0	0	0	0
W2	11	0	0	0	1
	10	0	0	0	0

W3W2H3H2'

green_indexing_type_2

			H3	H2	
		00	01	11	10
	00	0	0	0	0
W3	01	1	0	0	0
W2	11	0	0	0	0
	10	0	0	0	0

W3'W2H3'H2'

green_indexing_type_11

			H3	H2	
		00	01	11	10
	00	0	0	0	0
W3	01	0	1	0	0
W2	11	0	0	0	0
	10	0	0	0	0

W3'W2H3'H2

green_indexing_type_12

			H3	H2	
		00	01	11	10
	00	0	0	0	0
W3	01	0	0	0	0
W2	11	0	0	0	0
	10	0	0	0	1

W3W2'H3H2'

green_indexing_type_1

			H3	H2	
		00	01	11	10
	00	0	0	0	0
W3	01	0	0	0	0
W2	11	0	0	1	0
	10	0	0	0	0

W3W2H3H2

blue_indexing_all_on

			H3	H2	
		00	01	11	10
	00	1	1	0	1
W3	01	0	0	1	1
W2	11	0	0	0	0
	10	0	0	1	0

W3'W2'H3' + W3'H3H2' + W3'W2H3 +W3W2'H3H2

blue_indexing_type_13

			H3	H2	
		00	01	11	10
	00	0	0	1	0
W3	01	0	0	0	0
W2	11	0	0	0	0
	10	0	0	0	0

W3'W2'H3H2

big_subsystem_0101

blue_indexing_type_16

			H3	H2	
		00	01	11	10
	00	1	0	0	0
W3	01	0	0	0	0
W2	11	0	0	0	0
	10	0	0	0	0

W3'W2'H3'H2'

blue_indexing_type_4

			H3	H2	
		00	01	11	10
	00	0	0	0	0
W3	01	1	0	0	0
W2	11	0	0	0	0
	10	0	0	0	0

W3'W2H3'H2'

blue_indexing_type_15

			H3	H2	
		00	01	11	10
	00	0	0	0	0
W3	01	0	1	0	0
W2	11	0	0	0	0
	10	0	0	0	0

W3'W2H3'H2

blue_indexing_type_2

			H3	H2	
		00	01	11	10
	00	0	0	0	0
W3	01	0	0	0	0
W2	11	0	0	0	0
	10	0	0	0	1

W3W2'H3H2'

blue_indexing_type_11

			H3	H2	
		00	01	11	10
	00	0	0	0	0
W3	01	0	0	0	0
W2	11	0	0	0	0
	10	0	0	1	0

W3W2'H3H2

blue_indexing_all_on

			H3	H2	
		00	01	11	10
	00	0	0	0	0
W3	01	0	0	0	0
W2	11	0	1	1	1
	10	1	1	0	0

W3W2'H3' + W3H3'H2 + W3W2H3

green_indexing_type_3

			H3	H2	
		00	01	11	10
	00	0	0	0	0
W3	01	0	0	0	0
W2	11	1	0	0	0
	10	0	0	0	0

W3W2H3'H2'

purple_indexing_all_on

			H3	H2	
		00	01	11	10
	00	0	1	1	1
W3	01	0	0	1	1
W2	11	0	0	0	0
	10	0	0	0	0

W3'H3 + W3'W2'H2

big_subsystem_0110

purple_indexing_all_on

			H3	H2	
		00	01	11	10
	00	1	1	1	1
W3	01	1	1	1	1
W2	11	0	1	1	1
	10	1	1	1	1

W3' + H2 + H3 + W2'

blue_indexing_type_15

			H3	H2	
		00	01	11	10
	00	0	0	0	0
W3	01	0	0	0	0
W2	11	1	0	0	0
	10	0	0	0	0

W3W2H3'H2'

big_subsystem_1000

red_indexing_all_on

			H3	H2	
		00	01	11	10
	00	0	0	0	0
W3	01	0	0	0	0
W2	11	1	1	0	0
	10	1	0	0	0

W3W2H3' + W3H3'H2' = W3H3'(W2+H2')

red_indexing_type_1

			H3	H2	
		00	01	11	10
	00	0	0	0	0
W3	01	1	0	0	0
W2	11	0	0	0	0
	10	0	0	0	0

W3'W2H3'H2'

red_indexing_type_2

			H3	H2	
		00	01	11	10
	00	0	0	0	0
W3	01	0	0	0	0
W2	11	0	0	0	1
	10	0	1	0	0

W3W2'H3'H2 + W3W2H3H2'

red_indexing_type_3

			H3	H2	
		00	01	11	10
	00	0	0	0	0
W3	01	0	0	0	0
W2	11	0	0	1	0
	10	0	0	0	1

W3W2H3H2 + W3W2'H3H2'

yellow_indexing_all_on

			H3	H2	
		00	01	11	10
	00	1	1	0	0
W3	01	0	1	1	1
W2	11	0	0	0	0
	10	0	0	1	0

W3'W2'H3' + W3'H3'H2 + W3'W2H3 + W3W2'H3H2

yellow_indexing_type_4

			H3	H2	
		00	01	11	10
	00	0	0	0	1
W3	01	0	0	0	0
W2	11	0	0	0	0
	10	0	0	0	0

W3'W2'H3H2'

yellow_indexing_type_5

			H3	H2	
		00	01	11	10
	00	0	0	1	0
W3	01	0	0	0	0
W2	11	0	0	0	0
	10	0	0	0	0

W3'W2'H3H2

big_subsystem_1001

yellow_indexing_type_12

			H3	H2	
		00	01	11	10
	00	0	0	0	0
W3	01	1	0	0	0
W2	11	0	0	0	0
	10	0	0	0	0

W3'W2H3'H2'

yellow_indexing_type_1

			H3	H2	
		00	01	11	10
	00	0	0	0	0
W3	01	0	0	0	0
W2	11	0	0	0	1
	10	0	1	0	0

W3W2'H3'H2 + W3W2H3H2'

yellow_indexing_type_3

			H3	H2	
		00	01	11	10
	00	0	0	0	0
W3	01	0	0	0	0
W2	11	0	0	1	0
	10	0	0	0	0

W3W2H3H2

yellow_indexing_all_on

			H3	H2	
		00	01	11	10
	00	0	0	0	0
W3	01	0	0	0	0
W2	11	1	1	0	0
	10	1	0	0	0

W3W2H3' + W3H3'H2'

green_indexing_type_2

			H3	H2	
		00	01	11	10
	00	1	0	0	0
W3	01	0	0	0	0
W2	11	0	0	0	0
	10	0	0	0	0

W3'W2'H3'H2'

green_indexing_type_16

			H3	H2	
		00	01	11	10
	00	0	1	0	0
W3	01	0	0	0	0
W2	11	0	0	0	0
	10	0	0	0	0

W3'W2'H3'H2

green_indexing_type_4

			H3	H2	
		00	01	11	10
	00	0	0	0	0
W3	01	0	1	0	0
W2	11	0	0	0	0
	10	0	0	0	0

W3'W2H3'H2

green_indexing_type_5

			H3	H2	
		00	01	11	10
	00	0	0	0	0
W3	01	0	0	0	1
W2	11	0	0	0	0
	10	0	0	0	0

W3'W2H3H2'

green_indexing_type_12

			H3	H2	
		00	01	11	10
	00	0	0	0	0
W3	01	0	0	0	0
W2	11	0	0	0	0
	10	0	0	1	0

W3W2'H3H2

green_indexing_all_on

			H3	H2	
		00	01	11	10
	00	0	0	0	0
W3	01	0	0	0	0
W2	11	0	0	0	0
	10	0	0	0	1

W3W2'H3H2'

blue_indexing_all_on

			H3	H2	
		00	01	11	10
	00	0	0	1	1
W3	01	0	0	1	0
W2	11	0	0	0	0
	10	0	0	0	0

W3'W2'H3 + W3'H3H2

big_subsystem_1010

blue_indexing_all_on

			H3	H2	
		00	01	11	10
	00	1	0	0	0
W3	01	1	1	0	0
W2	11	0	1	1	1
	10	1	1	1	1

W3W2' + W3H2 + W3H3 + W3'W2H3'
+ W3'H3'H2'

blue_indexing_type_13

			H3	H2	
		00	01	11	10
	00	0	1	0	0
W3	01	0	0	0	0
W2	11	0	0	0	0
	10	0	0	0	0

W3'W2'H3'H2

blue_indexing_type_16

			H3	H2	
		00	01	11	10
	00	0	0	0	1
W3	01	0	0	0	0
W2	11	0	0	0	0
	10	0	0	0	0

W3'W2'H3H2'

blue_indexing_type_4

			H3	H2	
		00	01	11	10
	00	0	0	0	0
W3	01	0	0	0	1
W2	11	0	0	0	0
	10	0	0	0	0

W3'W2H3H2'

blue_indexing_type_9

			H3	H2	
		00	01	11	10
	00	0	0	0	0
W3	01	0	0	1	0
W2	11	0	0	0	0
	10	0	0	0	0

W3'W2H3H2

green_indexing_type_1

			H3	H2	
		00	01	11	10
	00	0	0	0	0
W3	01	0	0	0	0
W2	11	1	0	0	0
	10	0	0	0	0

W3W2H3'H2'

purple_indexing_all_on

			H3	H2	
		00	01	11	10
	00	0	0	1	0
W3	01	0	0	0	0
W2	11	0	0	0	0
	10	0	0	0	0

W3'W2'H3H2

big_subsystem_1011

purple_indexing_all_on

			H3	H2	
		00	01	11	10
	00	1	1	1	1
W3	01	1	1	1	1
W2	11	0	0	1	0
	10	0	0	1	1

W3' + W2'H3 + W3H3H2

blue_indexing_type_1

			H3	H2	
		00	01	11	10
	00	0	0	0	0
W3	01	0	0	0	0
W2	11	0	0	0	0
	10	1	0	0	0

W3W2'H3'H2'

blue_indexing_type_3

			H3	H2	
		00	01	11	10
	00	0	0	0	0
W3	01	0	0	0	0
W2	11	0	0	0	0
	10	0	1	0	0

W3W2'H3'H2

blue_indexing_type_2

			H3	H2	
		00	01	11	10
	00	0	0	0	0
W3	01	0	0	0	0
W2	11	0	1	0	0
	10	0	0	0	0

W3W2H3'H2

blue_indexing_type_16

			H3	H2	
		00	01	11	10
	00	0	0	0	0
W3	01	0	0	0	0
W2	11	0	0	0	1
	10	0	0	0	0

W3W2H3H2'

blue_indexing_all_on

			H3	H2	
		00	01	11	10
	00	0	0	0	0
W3	01	0	0	0	0
W2	11	1	0	0	0
	10	0	0	0	0

W3W2H3'H2'

big_subsystem_1100

red_indexing_all_on

			H3	H2	
		00	01	11	10
	00	1	1	0	1
W3	01	1	1	1	1
W2	11	1	1	1	1
	10	1	1	1	1

W3 + W2 + H3' + H2'

yellow_indexing_type_4

			H3	H2	
		00	01	11	10
	00	0	0	1	0
W3	01	0	0	0	0
W2	11	0	0	0	0
	10	0	0	0	0

W3'W2'H3H2

big_subsystem_1101

red_indexing_all_on

			H3	H2	
		00	01	11	10
	00	0	0	0	0
W3	01	0	0	0	0
W2	11	1	1	0	0
	10	1	0	0	0

W3W2H3' + W3H3'H2' = W3H3'(W2+H2')

red_indexing_type_2

			H3	H2	
		00	01	11	10
	00	0	0	0	0
W3	01	1	0	0	0
W2	11	0	0	0	0
	10	0	1	0	0

W3'W2H3'H2' + W3W2'H3'H2

red_indexing_type_3

			H3	H2	
		00	01	11	10
	00	1	0	0	0
W3	01	0	1	0	0
W2	11	0	0	0	0
	10	0	0	0	1

W3'W2'H3'H2' + W3'W2H3'H2 + W3W2'H3H2'

red_indexing_type_6

			H3	H2	
		00	01	11	10
	00	0	0	0	0
W3	01	0	0	0	0
W2	11	0	0	0	1
	10	0	0	0	0

W3W2H3H2'

yellow_indexing_all_on

			H3	H2	
		00	01	11	10
	00	0	1	0	1
W3	01	0	0	1	1
W2	11	0	0	1	0
	10	0	0	1	0

W3'W2'H3'H2 + W3'H3H2' + W3'W2H3 + W3H3H2

yellow_indexing_type2

			H3	H2	
		00	01	11	10
	00	0	0	1	0
W3	01	0	0	0	0
W2	11	0	0	0	0
	10	0	0	0	0

W3'W2'H3H2

big_subsystem_1110

yellow_indexing_all_on

			H3	H2	
		00	01	11	10
	00	0	0	0	0
W3	01	0	0	0	0
W2	11	1	1	0	0
	10	1	0	0	0

W3W2H3' + W3H3'H2'

yellow_indexing_type_2

			H3	H2	
		00	01	11	10
	00	0	0	0	0
W3	01	1	0	0	0
W2	11	0	0	0	1
	10	0	1	0	0

W3'W2H3'H2' + W3W2'H3'H2 + W3W2H3H2'

yellow_indexing_type_3

			H3	H2	
		00	01	11	10
	00	1	0	0	0
W3	01	0	1	0	0
W2	11	0	0	1	0
	10	0	0	0	1

W3'W2'H3'H2' + W3'W2H3'H2 + W3W2H3H2 + W3W2'H3H2'

green_indexing_type_10

			H3	H2	
		00	01	11	10
	00	0	1	0	0
W3	01	0	0	0	0
W2	11	0	0	0	0
	10	0	0	0	0

W3'W2'H3'H2

green_indexing_type_2

			H3	H2	
		00	01	11	10
	00	0	0	0	0
W3	01	0	0	0	1
W2	11	0	0	0	0
	10	0	0	1	0

W3'W2H3H2' + W3W2'H3H2

green_indexing_type_3

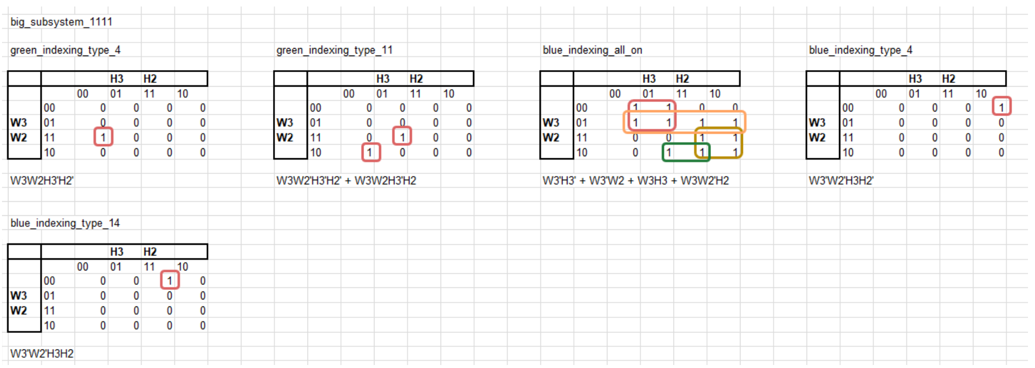
			H3	H2	
		00	01	11	10
	00	0	0	0	1
W3	01	0	0	1	0
W2	11	0	0	0	0
	10	0	0	0	0

W3'W2H3H2 + W3'W2'H3H2'

blue_indexing_type_all_on

			H3	H2	
		00	01	11	10
	00	0	0	1	0
W3	01	0	0	0	0
W2	11	0	0	0	0
	10	0	0	0	0

W3'W2'H3H2



big_subsystem_1111

green_indexing_type_4

		H3 H2			
		00	01	11	10
	00	0	0	0	0
W3	01	0	0	0	0
W2	11	1	0	0	0
	10	0	0	0	0

W3W2H3'H2'

green_indexing_type_11

		H3 H2			
		00	01	11	10
	00	0	0	0	0
W3	01	0	0	0	0
W2	11	0	1	0	0
	10	1	0	0	0

W3W2'H3'H2' + W3W2H3'H2

blue_indexing_all_on

		H3 H2			
		00	01	11	10
	00	1	1	0	0
W3	01	1	1	1	1
W2	11	0	0	1	1
	10	0	1	1	1

W3'H3' + W3'W2 + W3H3 + W3W2'H2

blue_indexing_type_4

		H3 H2			
		00	01	11	10
	00	0	0	0	1
W3	01	0	0	0	0
W2	11	0	0	0	0
	10	0	0	0	0

W3'W2'H3H2'

blue_indexing_type_14

		H3 H2			
		00	01	11	10
	00	0	0	1	0
W3	01	0	0	0	0
W2	11	0	0	0	0
	10	0	0	0	0

W3'W2'H3H2

Figure 1.5.1. All ***small indexing*** KMaps and SOPs from ***big_subsystem_0000*** to ***big_subsystem_1111***

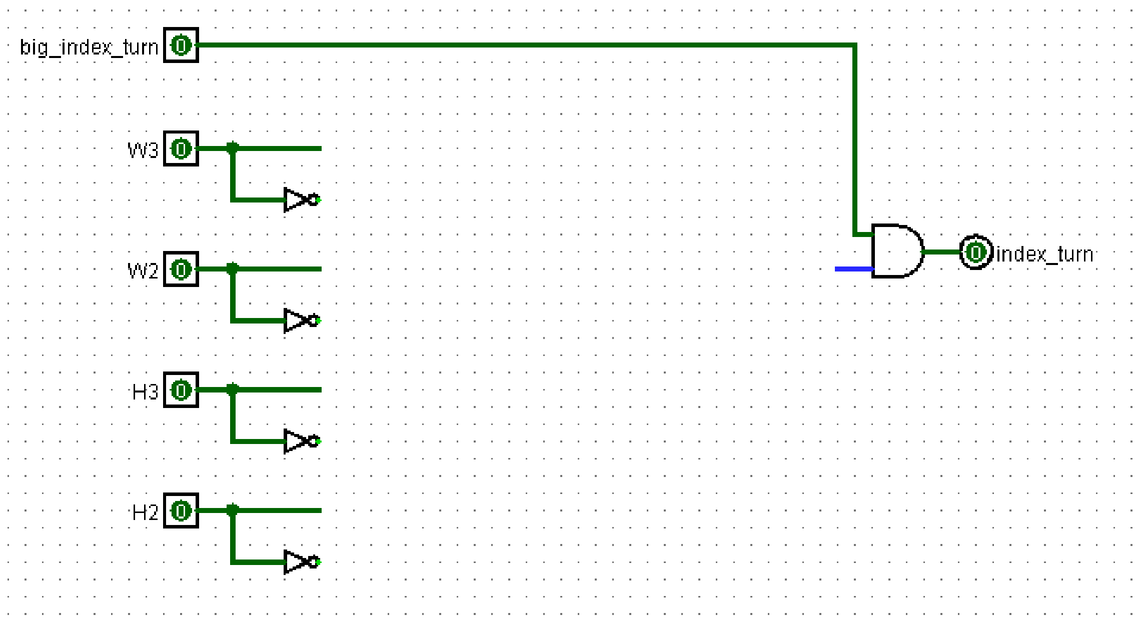


Figure 1.5.2. Color ***small indexing*** template

1.6 Underlying Patterns and Different KMap Types

Since for each ***small subsystem*** found in each ***big subsystem*** contain similar 4 by 4 KMaps (i.e. can also be found in other subsystems too), we decided to implement separate circuits for these 16 distinct KMap patterns to be reused as well in other circuits (see Figure 1.6.1).

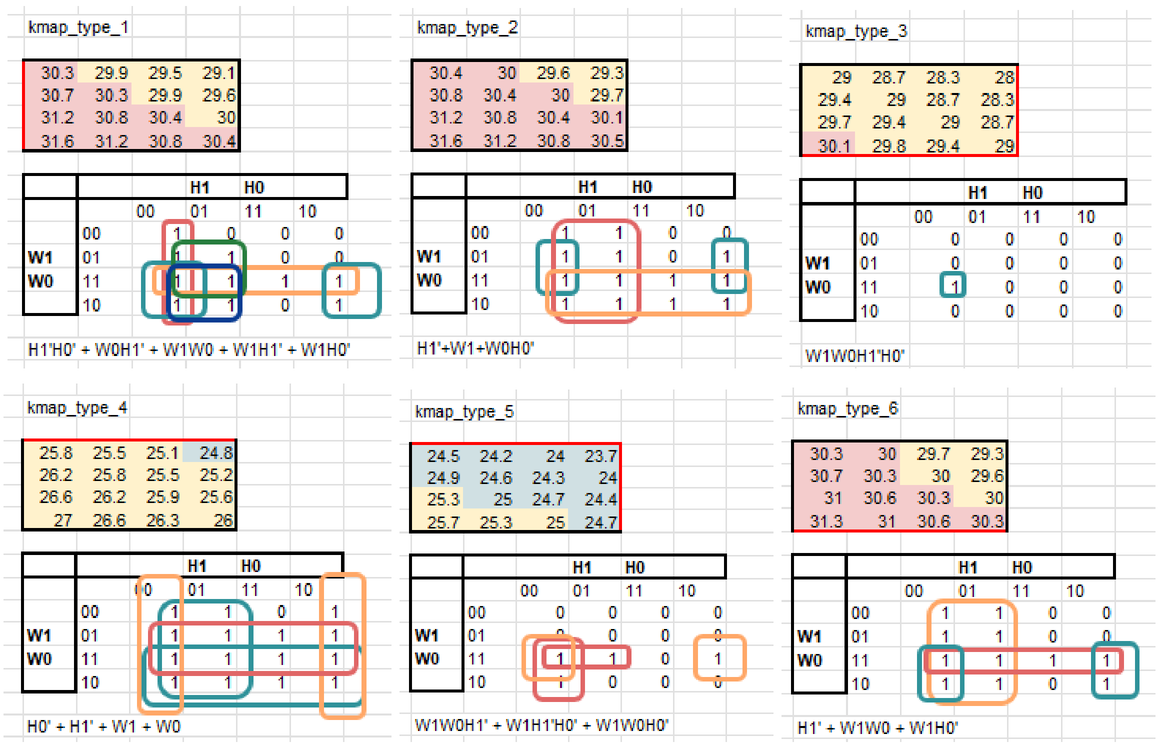
Note that for each KMap type, we used a KMap template (see Figure 1.6.2). It consists of the following inputs:

- ***index_turn*** – depends on ***small indexing*** logic in whether a KMap type should be activated or not
- ***W1, W0*** – these are the 2 least significant bits of ***Weight***
- ***H1, H0*** – these are the 2 least significant bits of ***Height***

It also is consists of the following outputs:

- ***on_color*** – this refers to a color ***itself*** that is ***on*** with respect to the KMap type SOP
- ***off_color*** – this refers to the ***other*** color that is ***on*** with respect to the KMap type SOP

For example, *kmap_type_1* has red as ***on_color*** and yellow as ***off_color***.



kmap_type_1

30.3	29.9	29.5	29.1
30.7	30.3	29.9	29.6
31.2	30.8	30.4	30
31.6	31.2	30.8	30.4

		H1 H0			
		00	01	11	10
	00	1	0	0	0
W1	01	1	1	0	0
W0	11	1	1	1	1
	10	1	1	0	1

H1'H0' + W0H1' + W1W0 + W1H1' + W1H0'

kmap_type_2

30.4	30	29.6	29.3
30.8	30.4	30	29.7
31.2	30.8	30.4	30.1
31.6	31.2	30.8	30.5

		H1 H0			
		00	01	11	10
	00	1	1	0	0
W1	01	1	1	0	1
W0	11	1	1	1	1
	10	1	1	1	1

H1'+W1+W0H0'

kmap_type_3

29	28.7	28.3	28
29.4	29	28.7	28.3
29.7	29.4	29	28.7
30.1	29.8	29.4	29

		H1 H0			
		00	01	11	10
	00	0	0	0	0
W1	01	0	0	0	0
W0	11	1	0	0	0
	10	0	0	0	0

W1W0H1'H0'

kmap_type_4

25.8	25.5	25.1	24.8
26.2	25.8	25.5	25.2
26.6	26.2	25.9	25.6
27	26.6	26.3	26

		H1 H0			
		00	01	11	10
	00	1	1	0	1
W1	01	1	1	1	1
W0	11	1	1	1	1
	10	1	1	1	1

H0' + H1' + W1 + W0

kmap_type_5

24.5	24.2	24	23.7
24.9	24.6	24.3	24
25.3	25	24.7	24.4
25.7	25.3	25	24.7

		H1 H0			
		00	01	11	10
	00	0	0	0	0
W1	01	0	0	0	0
W0	11	1	1	0	1
	10	1	0	0	0

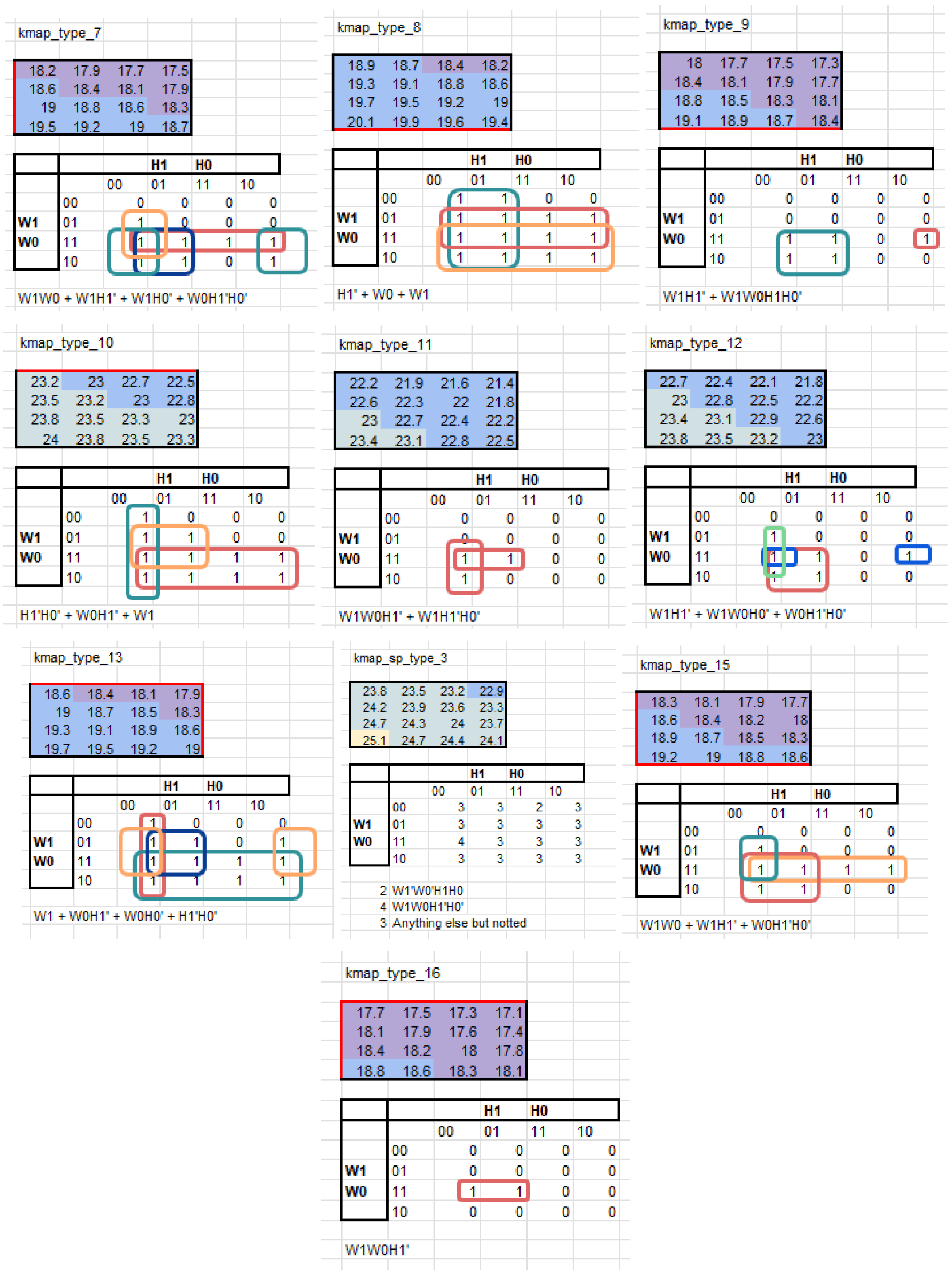
W1W0H1' + W1H1'H0' + W1W0H0'

kmap_type_6

30.3	30	29.7	29.3
30.7	30.3	30	29.6
31	30.6	30.3	30
31.3	31	30.6	30.3

		H1 H0			
		00	01	11	10
	00	1	1	0	0
W1	01	1	1	0	0
W0	11	1	1	1	1
	10	1	1	0	1

H1' + W1W0 + W1H0'



kmap_type_7

18.2	17.9	17.7	17.5
18.6	18.4	18.1	17.9
19	18.8	18.6	18.3
19.5	19.2	19	18.7

		H1 H0			
		00	01	11	10
	00	0	0	0	0
W1	01	1	0	0	0
W0	11	1	1	1	1
	10	1	1	0	1

W1W0 + W1H1' + W1H0' + W0H1'H0'

kmap_type_8

18.9	18.7	18.4	18.2
19.3	19.1	18.8	18.6
19.7	19.5	19.2	19
20.1	19.9	19.6	19.4

		H1 H0			
		00	01	11	10
	00	1	1	0	0
W1	01	1	1	1	1
W0	11	1	1	1	1
	10	1	1	1	1

H1' + W0 + W1

kmap_type_9

18	17.7	17.5	17.3
18.4	18.1	17.9	17.7
18.8	18.5	18.3	18.1
19.1	18.9	18.7	18.4

		H1 H0			
		00	01	11	10
	00	0	0	0	0
W1	01	0	0	0	0
W0	11	1	1	0	1
	10	1	1	0	0

W1H1' + W1W0H1H0'

kmap_type_10

23.2	23	22.7	22.5
23.5	23.2	23	22.8
23.8	23.5	23.3	23
24	23.8	23.5	23.3

		H1 H0			
		00	01	11	10
	00	1	0	0	0
W1	01	1	1	0	0
W0	11	1	1	1	1
	10	1	1	1	1

H1'H0' + W0H1' + W1

kmap_type_11

22.2	21.9	21.6	21.4
22.6	22.3	22	21.8
23	22.7	22.4	22.2
23.4	23.1	22.8	22.5

		H1 H0			
		00	01	11	10
	00	0	0	0	0
W1	01	0	0	0	0
W0	11	1	1	0	0
	10	1	0	0	0

W1W0H1' + W1H1'H0'

kmap_type_12

22.7	22.4	22.1	21.8
23	22.8	22.5	22.2
23.4	23.1	22.9	22.6
23.8	23.5	23.2	23

		H1 H0			
		00	01	11	10
	00	0	0	0	0
W1	01	1	0	0	0
W0	11	1	1	0	1
	10	1	1	0	0

W1H1' + W1W0H0' + W0H1'H0'

kmap_type_13

18.6	18.4	18.1	17.9
19	18.7	18.5	18.3
19.3	19.1	18.9	18.6
19.7	19.5	19.2	19

		H1 H0			
		00	01	11	10
	00	1	0	0	0
W1	01	1	1	0	1
W0	11	1	1	1	1
	10	1	1	1	1

W1 + W0H1' + W0H0' + H1'H0'

kmap_sp_type_3

23.8	23.5	23.2	22.9
24.2	23.9	23.6	23.3
24.7	24.3	24	23.7
25.1	24.7	24.4	24.1

		H1 H0			
		00	01	11	10
	00	3	3	2	3
W1	01	3	3	3	3
W0	11	4	3	3	3
	10	3	3	3	3

2	W1'W0'H1H0
4	W1W0H1'H0'
3	Anything else but notted

kmap_type_15

18.3	18.1	17.9	17.7
18.6	18.4	18.2	18
18.9	18.7	18.5	18.3
19.2	19	18.8	18.6

		H1 H0			
		00	01	11	10
	00	0	0	0	0
W1	01	1	0	0	0
W0	11	1	1	1	1
	10	1	1	0	0

W1W0 + W1H1' + W0H1'H0'

kmap_type_16

17.7	17.5	17.3	17.1
18.1	17.9	17.6	17.4
18.4	18.2	18	17.8
18.8	18.6	18.3	18.1

		H1 H0			
		00	01	11	10
	00	0	0	0	0
W1	01	0	0	0	0
W0	11	1	1	0	0
	10	0	0	0	0

W1W0H1'

Figure 1.6.1. All 16 distinct KMap types

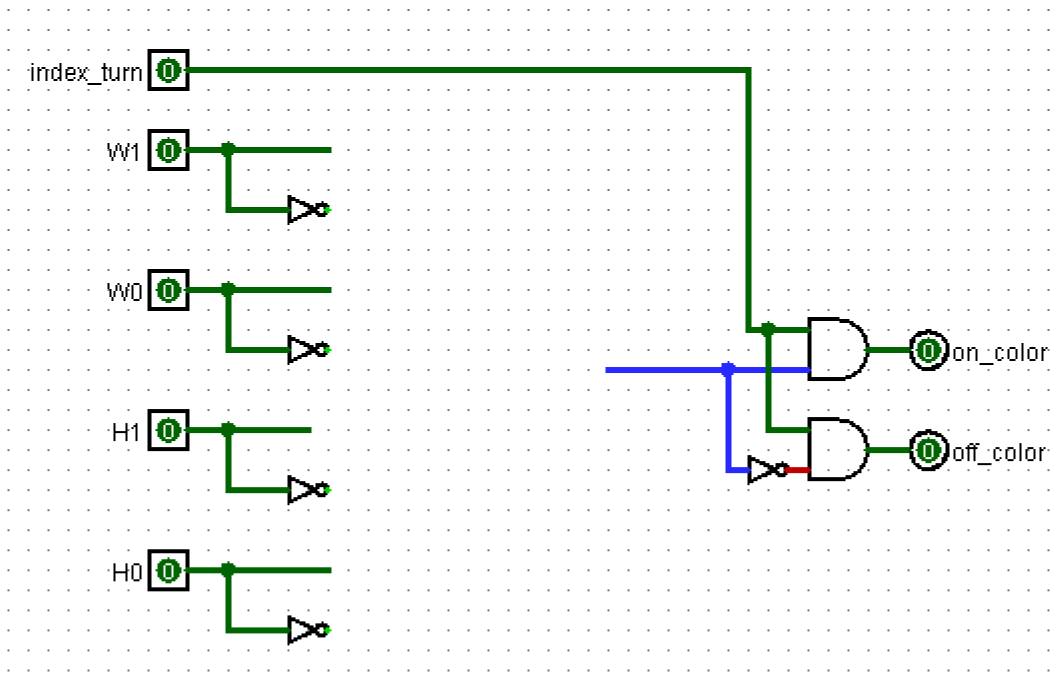


Figure 1.6.2. KMap template

1.7 The Circuit Implementation

Each of the 16 ***big subsystems*** has their own .circ file (e.g. *big_subsystem_0000.circ*) consisting of the following:

- **<color>_*indexing*_<kmap_type>** – this refers to the ***big subsystem***'s ***small indexing*** circuit implementation
- **kmap_type_<number>** – this refers to a KMap type circuit implementation
- **big_subsystem_<number>** – this refers to the ***big subsystem*** itself containing the logic of ***big indexing***, ***small indexing***, and ***kmap_type*** all together

After implementing each ***big subsystem***, it will then be imported in the *main.circ* and have them wired to their corresponding inputs and outputs.

We decided to have their own .circ file due to it being modular, and so that we can also split the work per member.

Additionally, if you were to download *main.circ*, make sure to download all other ***big subsystem*** files as well (e.g. *big_subsystem_0000.circ* to *big_subsystem_1111.circ*) and import them as Logisim library (if not yet automatically imported).

Last note, *main_demo* circuit in the *main.circ* file is ***not included*** in the Logisim project and is for ***demonstration purposes*** only.

1.8 How Everything Works

Each ***big subsystem*** and ***small subsystem*** contains a check, in the form of an *AND* gate, on whether or not its output should proceed for that combination of inputs. Because of this check, it is guaranteed that there will not be any conflicts since only one subsystem at a time will be active. By having each system have its own indexing, it prevents one subsystem from modifying the output of another.

The *main.circ* file abstracts the functionality of the BMI classification circuit into one (1) IC. It depends on the files *big_subsystem_0000* to *big_subsystem_1111*. If you're going to open main.circ, ***make sure that the big subsystem circuit files are in the same file directory as*** *main.circ*.

Links for the demonstration are provided below:

- Original demonstration video: cs20_logisim_demo.mp4
- Compressed version, to be uploaded to UVLê: cs20_logisim_demo_50mb.mp4

1.9 What Each File Submission Means

1. *main.circ*: The main Logisim file.

2. *big_subsystem_0000.circ* – *big_subsystem_1111.circ*: An implementation of *big_subsystem_0000*, *big_subsystem_0001*, *big_subsystem_0010*, . . . , *big_subsystem_1111* in Logisim as described in Figure 1.7 The Circuit Implementation.

3. *cs20project1.csv* – A table that contains every combination of the inputs *W5, W4, W3, W2, W1, W0, H5, H4, H3, H2, H1, H0*, and its corresponding output, either *x1, x2, x3, x4, or x5*. The outputs correspond to ***Underweight***, ***Normal Weight***, ***Overweight***, ***Obese Class I***, and ***Obese Class II*** respectively.

2 Physical Component

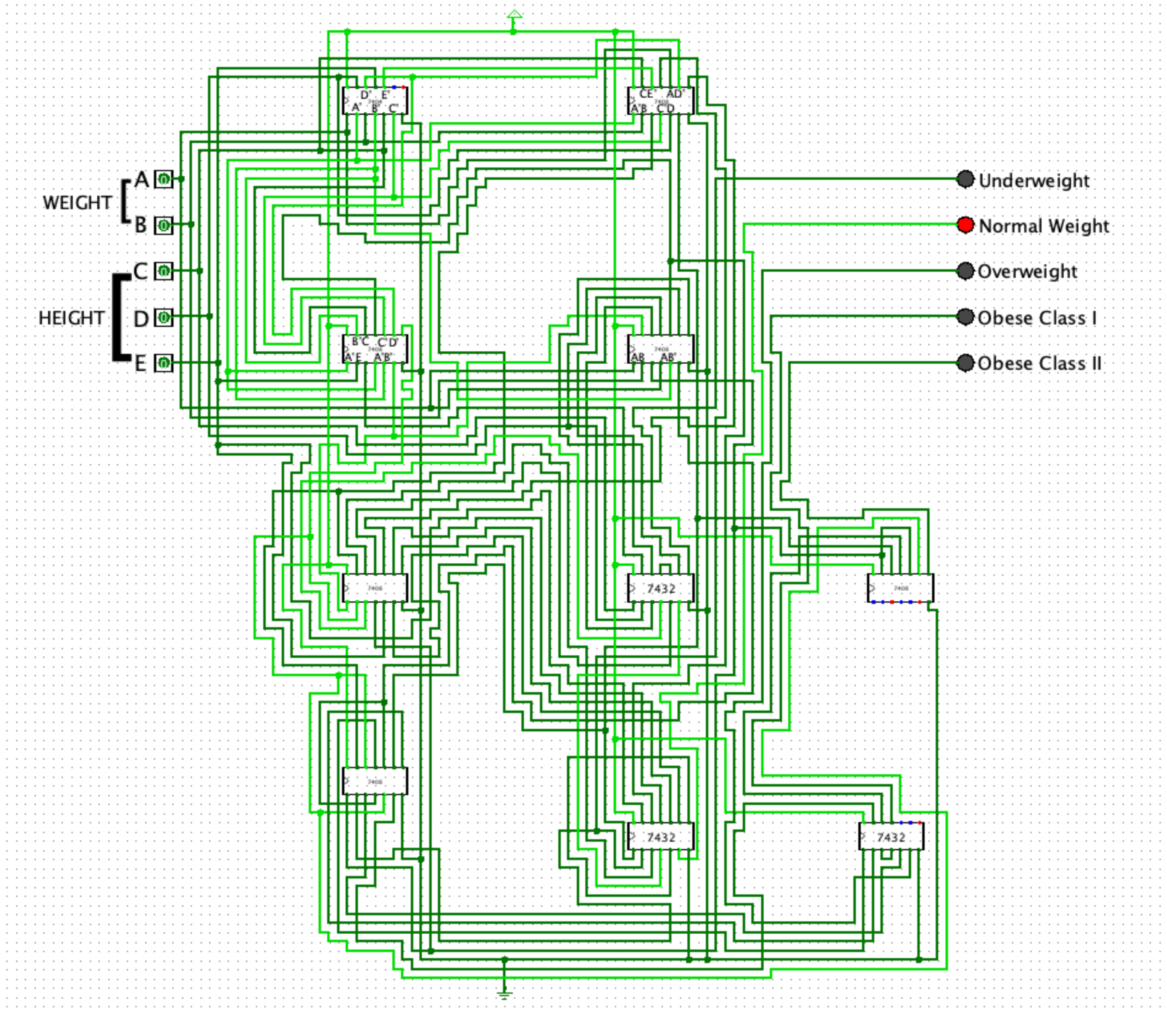
2.1 Specifications Recap

The physical implementation focuses on a <u>5-bit BMI calculator circuit</u> using combinational logic ICs. The system uses:

- **2 bits** for ***Weight*** (*W0*, *W1*) and **3 bits** for ***Height*** (*H0*, *H1*, *H2*).
- Combined in the input order as ***W0W1H0H1H2***, labelled as ***ABCDE*** for simplicity.

The system classifies the BMI into ***5 categories***, each represented by an LED. With 5 bits of input, there are 32 possible combinations.

2.2 LogiSim Diagram



The circuit design is illustrated above, with the following key components:

1. **Input Switches**
- Located on the left side, the switches are labeled **A, B, C, D, E,** oriented vertically from top to bottom. These represent the user-inputted weight and height values, with A as the *Most Significant Bit (MSB)* and E as the *Least Significant Bit (LSB)* of the 5-bit input.

2. **Circuit Body**
- The middle section contains the wiring and ICs responsible for processing the input and determining the corresponding BMI category. More specifically, the circuit consists of **6 74LS08 ICs** (AND), **3 74LS32 ICs** (OR), and **1 74LS04 IC** (NOT).

3. **Output LEDs**
- On the right side, there are five LEDs labeled ***Underweight, Normal Weight, Overweight, Obese Class I, and Obese Class II,*** arranged vertically from top to bottom.

When a user inputs a weight and height combination, the appropriate LED lights up, indicating the BMI category corresponding to the input.

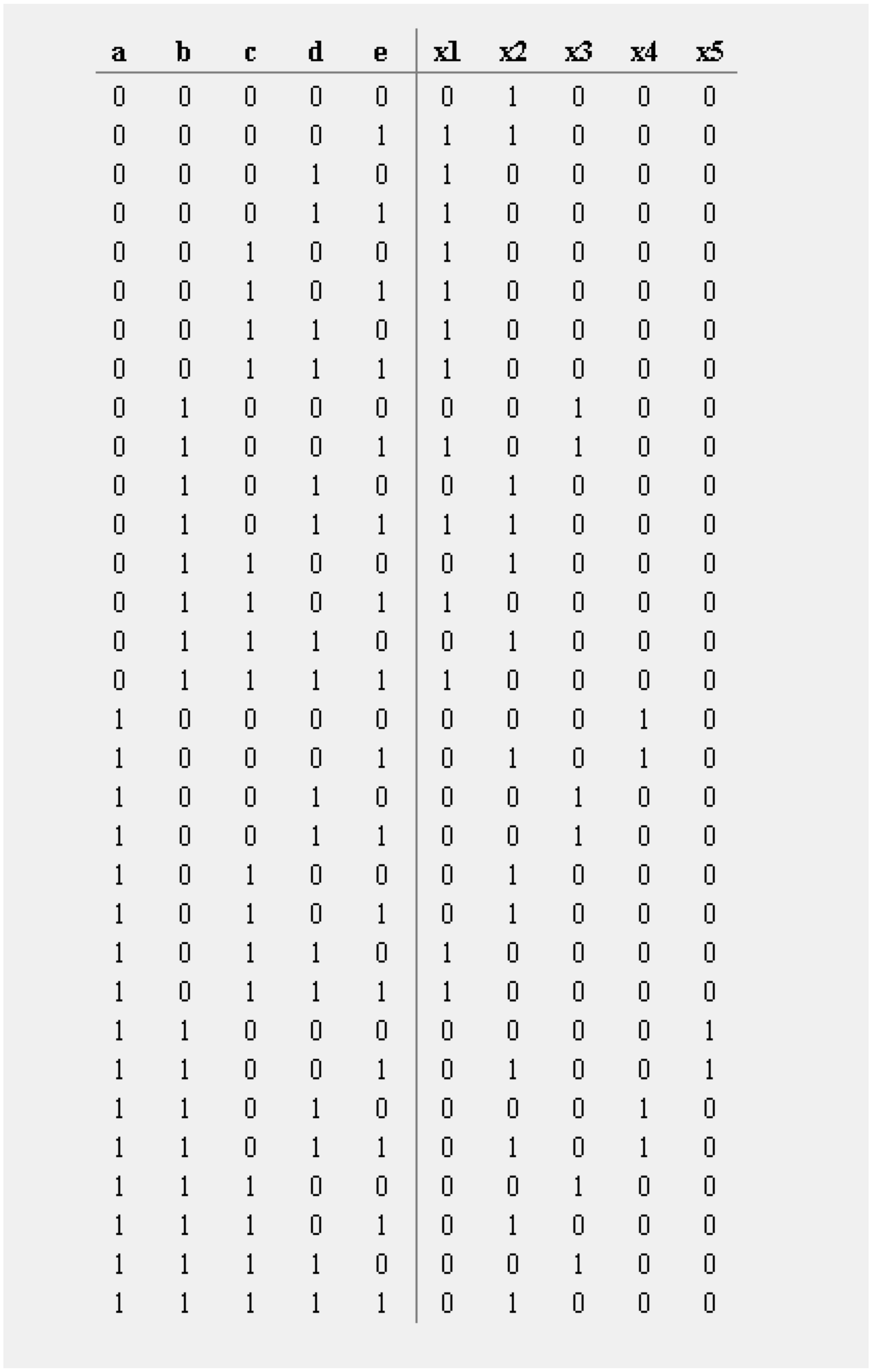
2.3 Truth Tables

Table 2.3.1. Main Table (Summary)

Weight (2-bit)	**Height (3-bit)**	**Underw. (LED 1)**	**Normal (LED 2)**	**Overw. (LED 3)**	**Obese I (LED 4)**	**Obese II (LED 5)**
00	**000**	0	1	0	0	0
00	**001**	X	X	X	X	X
00	**010**	1	0	0	0	0
00	**011**	X	X	X	X	X
00	**100**	1	0	0	0	0
00	**101**	1	0	0	0	0
00	**110**	X	X	X	X	X
00	**111**	1	0	0	0	0

01	**000**	0	0	1	0	0
01	**001**	X	X	X	X	X
01	**010**	0	1	0	0	0
01	**011**	X	X	X	X	X
01	**100**	0	1	0	0	0
01	**101**	1	0	0	0	0
01	**110**	X	X	X	X	X
01	**111**	1	0	0	0	0
10	**000**	0	0	0	1	0
10	**001**	X	X	X	X	X
10	**010**	0	0	1	0	0
10	**011**	X	X	X	X	X
10	**100**	0	1	0	0	0
10	**101**	0	1	0	0	0
10	**110**	X	X	X	X	X
10	**111**	1	0	0	0	0
11	**000**	0	0	0	0	1
11	**001**	X	X	X	X	X
11	**010**	0	0	0	1	0
11	**011**	X	X	X	X	X
11	**100**	0	0	1	0	0
11	**101**	0	1	0	0	0
11	**110**	X	X	X	X	X
11	**111**	0	1	0	0	0

Table 2.3.2. Logisim Output



a	b	c	d	e	x1	x2	x3	x4	x5
0	0	0	0	0	0	1	0	0	0
0	0	0	0	1	1	1	0	0	0
0	0	0	1	0	1	0	0	0	0
0	0	0	1	1	1	0	0	0	0
0	0	1	0	0	1	0	0	0	0
0	0	1	0	1	1	0	0	0	0
0	0	1	1	0	1	0	0	0	0
0	0	1	1	1	1	0	0	0	0
0	1	0	0	0	0	0	1	0	0
0	1	0	0	1	1	0	1	0	0
0	1	0	1	0	0	1	0	0	0
0	1	0	1	1	1	1	0	0	0
0	1	1	0	0	0	1	0	0	0
0	1	1	0	1	1	0	0	0	0
0	1	1	1	0	0	1	0	0	0
0	1	1	1	1	1	0	0	0	0
1	0	0	0	0	0	0	0	1	0
1	0	0	0	1	0	1	0	1	0
1	0	0	1	0	0	0	1	0	0
1	0	0	1	1	0	0	1	0	0
1	0	1	0	0	0	1	0	0	0
1	0	1	0	1	0	1	0	0	0
1	0	1	1	0	1	0	0	0	0
1	0	1	1	1	1	0	0	0	0
1	1	0	0	0	0	0	0	0	1
1	1	0	0	1	0	1	0	0	1
1	1	0	1	0	0	0	0	1	0
1	1	0	1	1	0	1	0	1	0
1	1	1	0	0	0	0	1	0	0
1	1	1	0	1	0	1	0	0	0
1	1	1	1	0	0	0	1	0	0
1	1	1	1	1	0	1	0	0	0

2.4 K-Maps and Minimal SOPs

The Boolean expressions for each LED were derived using **Karnaugh maps (K-Maps)** to minimize the **Sum-of-Products (SOP)** expressions. Each K-Map considers the variable C (the third bit of the height input) as the **pivot point**, allowing the separation of the logic into two cases: **C = 0 and C = 1**. This simplification strategy ensures a structured and systematic derivation process.

Table 2.4.1 & 2.4.2. LED 1 K-Map (C = 0 and C = 1)

LED 1 (C is 0)		**DE**				**LED 1 (C is 1)**		**DE**			
		00	**01**	**11**	**10**			**00**	**01**	**11**	**10**
AB	**00**	0	X	X	1	**AB**	**00**	1	1	1	X
	01	0	X	X	0		**01**	0	1	1	X
	11	0	X	X	0		**11**	0	0	0	X
	10	0	X	X	0		**10**	0	0	1	X

Table 2.4.3 & 2.4.4. LED 2 K-Map (C = 0 and C = 1)

LED 2 (C is 0)		**DE**				**LED 2 (C is 1)**		**DE**			
		00	**01**	**11**	**10**			**00**	**01**	**11**	**10**
AB	**00**	1	X	X	0	**AB**	**00**	0	0	0	X
	01	0	X	X	1		**01**	1	0	0	X
	11	0	X	X	0		**11**	0	1	1	X
	10	0	X	X	0		**10**	1	1	0	X

Table 2.4.5 & 2.4.6. LED 3 K-Map (C = 0 and C = 1)

LED 3 (C is 0)		**DE**				**LED 3 (C is 1)**		**DE**			
		00	**01**	**11**	**10**			**00**	**01**	**11**	**10**
AB	**00**	0	X	X	0	**AB**	**00**	0	0	0	X
	01	1	X	X	0		**01**	0	0	0	X
	11	0	X	X	0		**11**	1	0	0	X
	10	0	X	X	1		**10**	0	0	0	X

Table 2.4.7 & 2.4.8. LED 4 K-Map (C = 0 and C = 1)

LED 4 (C is 0)		**DE**				**LED 4 (C is 1)**		**DE**			
		00	**01**	**11**	**10**			**00**	**01**	**11**	**10**
AB	**00**	0	X	X	0	**AB**	**00**	0	0	0	X
	01	0	X	X	0		**01**	0	0	0	X
	11	0	X	X	1		**11**	0	0	0	X
	10	1	X	X	0		**10**	0	0	0	X

Table 2.4.9 & 2.4.10. LED 5 K-Map (C = 0 and C = 1)

LED 5 (C is 0)		**DE**				**LED 5 (C is 1)**		**DE**			
		00	**01**	**11**	**10**			**00**	**01**	**11**	**10**
AB	**00**	0	X	X	0	**AB**	**00**	0	0	0	X
	01	0	X	X	0		**01**	0	0	0	X
	11	1	X	X	0		**11**	0	0	0	X
	10	0	X	X	0		**10**	0	0	0	X

Table 2.4.11. Derived Boolean Expressions From K-Maps

LED	BMI Category	SOP Boolean Expression
x1	**Underweight**	A'E + A'B'D + A'B'C + B'CD
x2	**Normal Weight**	A'B'C'D' + A'BC'D + A'BCE' + AD'E + AB'CD' + ABE
x3	**Overweight**	A'BC'D' + AB'C'D + ABCE'
x4	**Obese Class I**	AB'C'D' + ABC'D
x5	**Obese Class II**	ABC'D'

The SOPs were further optimized by recognizing **10 common AND expressions** to reduce complexity. The simplified expressions are as follows:

Table 2.4.12. Simplified Boolean Expressions

LED	BMI Category	SOP Boolean Expression
x1	**Underweight**	(A'E) + (A'B')(C + D) + (B'C)D
x2	**Normal Weight**	(A'B')(C'D') + (A'B)((C'D) + (CE'))+ (AD')(E + (B'C)) + (AB)E
x3	**Overweight**	(A'B)(C'D') + (AB')(C'D) + (AB)(CE')
x4	**Obese Class I**	(AB')(C'D') + (AB)(C'D)
x5	**Obese Class II**	(AB)(C'D')

Table 2.4.13. Common AND Expressions

A’E	1
A’B’	3
B’C	2
C’D’	4
A’B	3
C’D	3
CE’	2
AD’	2
AB	4
AB’	2
TOTAL	26

2.5 Input Variable C as the “Pivot Point”

The decision to use C (the third bit of the height input) as the **"pivot point"** was driven by several ***optimization factors***:

[REDUCTION IN COMPLEXITY]

- Selecting C as the pivot point eliminates the need for 5-variable expressions in all SOPs. The largest expressions across all SOPs are reduced to 4-variable groupings, significantly simplifying the circuit design.
- Notably, this choice ensures there are no isolated 1s (1x1 groupings) in the minimal SOP groupings, further streamlining the logic.

[OPTIMIZED GROUPINGS]

- With C as the pivot point, the smallest groupings in the Karnaugh maps are limited to 2x1 or 1x2 rectangles, facilitating more efficient Boolean simplifications.

[ENHANCED CIRCUIT ORGANIZATION]

- Using C reveals multiple shared terms across the 5 SOPs (see Table 2.4.13). These commonalities make it easier to combine and organize the logic into subcircuits, improving the overall design's modularity and efficiency.

2.6 Selection of 3-Bit Input Combinations

The selection of the 3-bit height input combinations 000, 010, 100, 101, and 111 was primarily guided by the goal of optimizing the K-Maps and minimizing circuit complexity. Among all possible combinations, these specific values were chosen for the following reasons:

[AVOIDING ISOLATED 1S IN K-MAPS]

- The chosen combinations ensured that the K-Maps for all BMI categories avoided isolated 1s (1x1 groupings). This was critical for simplifying the Boolean expressions, as larger groupings (e.g., 2x1 or 2x2) lead to fewer terms in the final SOP expressions.

[MAXIMIZING GROUPING EFFICIENCY]

- By strategically selecting these combinations, we maximized the opportunities for grouping adjacent 1s in the K-Maps. This resulted in minimal overlap and reduced the number of terms required for each LED's logic.

[SIGNIFICANT REDUCTION IN IC CONSUMPTION]

- The optimized input combinations significantly reduced the number of ICs required to implement the circuit. Instead of originally using **24 ICs**, the design was simplified to only **10 ICs**, resulting in a more efficient and cost-effective physical implementation.

[ALIGNMENT WITH REALISTIC INPUTS]

- The selected patterns also aligned well with the range of valid height inputs defined in the project. This ensured that the logic circuit covered meaningful cases while avoiding unnecessary combinations.