

## Lab 1 : Implementing Combinational Logic in the MAX10 FPGA

### Part 1:

- I decided to use the key buttons as input to the design. If both key buttons were off or pressed, then the LEDR[1] would be off. However, if one key button was pressed, while the other button was not pressed, then the LEDR[1] would turn on. Figure 1, LEDR1 is assigned to the combinational logic for this design.

```
assign LEDR[1] = (~KEY[0]&KEY[1]) | (~KEY[1]&KEY[0]);
```

**Figure 1: part1.v**

- A test bench module was given in the lab manual, and I modified the count register as a sequence of 2 bits, and is assigned to the key button inputs. There are 4 possible inputs and in Figure 2 displays the expected outputs.

```
{sim:/tb_part1/LEDR[1]}
VSIM 8> run
# in = 00, out = 0
run
# in = 01, out = 1
run
# in = 10, out = 1
VSIM 9> run
# in = 11, out = 0
```

**Figure 2: printed tb\_part1.v input and outputs**

### Part 2:

- Figure 3 displays the 16 different possible inputs and outputs of this design. We took advantage of Karnaugh maps to help produce the sum of products for each output. Figure 4 shows the 10 calculated combinational logic equations.
- The testbench was similar, except for adjusting the count to be assigned to the switch inputs and setting the display output to the 2 seven segment displays. Figure 5 shows the expected outputs.

					Hex[0]								Hex[1]							
	sw[3]	sw[2]	sw[1]	sw[0]	[6]	[5]	[4]	[3]	[2]	[1]	[0]	[6]	[5]	[4]	[3]	[2]	[1]	[0]		
0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0		
1	0	0	0	1	1	1	1	1	0	0	1	1	0	0	0	0	0	0		
2	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0		
3	0	0	1	1	0	1	1	0	0	0	0	1	0	0	0	0	0	0		
4	0	1	0	0	0	0	1	1	0	0	1	1	0	0	0	0	0	0		
5	0	1	0	1	0	0	1	0	0	1	0	1	0	0	0	0	0	0		
6	0	1	1	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0		
7	0	1	1	1	1	1	1	1	0	0	0	1	0	0	0	0	0	0		
8	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0		
9	1	0	0	1	0	0	1	1	0	0	0	1	0	0	0	0	0	0		
10	1	0	1	0	1	0	0	0	0	0	0	1	1	1	1	0	0	1		
11	1	0	1	1	1	1	1	1	0	0	1	1	1	1	1	0	0	1		
12	1	1	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	1		
13	1	1	0	1	0	1	1	0	0	0	0	1	1	1	1	0	0	1		
14	1	1	1	0	0	0	1	1	0	0	1	1	1	1	1	0	0	1		
15	1	1	1	1	1	0	0	1	0	0	1	0	1	1	1	1	0	0		

Figure 3: Truth table

$\Sigma(0,1,7,10,11)$

	$S_3 S_2$	$S_1 S_0$	00	01	11	10
00			1	1	0	0
01			0	0	1	0
11			0	0	0	0
10			0	0	1	1

$$S_3' S_2' S_1' + S_3 S_2' S_1 + S_3' S_2 S_1 S_0 = \text{Hexo}[6]$$

$\Sigma(1,2,3,7,11,12,13)$

	$S_3 S_2$	$S_1 S_0$	00	01	11	10
00			0	1	1	1
01			0	0	1	0
11			1	1	0	0
10			0	0	1	0

$$S_3' S_2' S_0 + S_3' S_2' S_1 + S_3' S_1 S_0 + S_2' S_1 S_0 + S_3 S_2 S_1' = \text{Hexo}[6]$$

$\Sigma(1,3,4,5,7,9,11,13,14,15)$

	$S_3 S_2$	$S_1 S_0$	00	01	11	10
00			0	1	1	0
01			1	1	1	0
11			0	1	1	1
10			0	1	1	0

$$S_0 + S_3 S_2 S_1 + S_3' S_2 S_1' = \text{Hexo}[4]$$

$\Sigma(1,4,7,9,11,14)$

	$S_3 S_2$	$S_1 S_0$	00	01	11	10
00			0	1	0	0
01			1	0	1	0
11			0	0	0	1
10			0	1	1	0

$$S_3 S_2' S_0 + S_2' S_1' S_0 + S_3' S_2 S_1' S_0 + S_3' S_2 S_1 S_0 + S_3' S_2 S_1 S_0' + S_3 S_2 S_1 S_0' = \text{Hexo}[3]$$

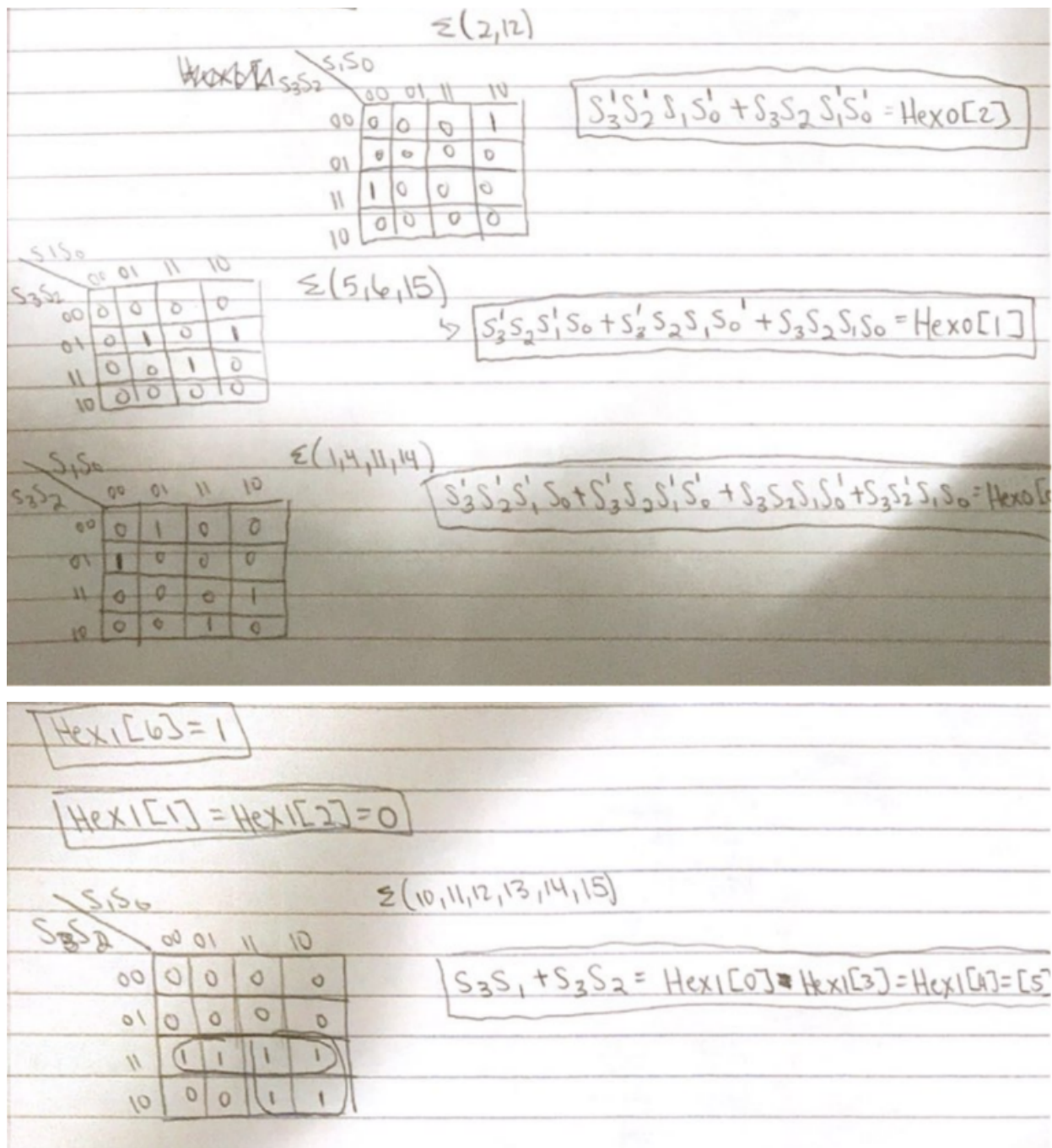


Figure 4: Karnaugh maps and output equations

```

sim:/tb_part2/HEX1
VSIM 5> run
# in = 0000, HEX1 = z1000000, HEX0 = z1000000
run
# in = 0001, HEX1 = z1000000, HEX0 = z1111001
run
# in = 0010, HEX1 = z1000000, HEX0 = z0100100
run
# in = 0011, HEX1 = z1000000, HEX0 = z0110000
run
# in = 0100, HEX1 = z1000000, HEX0 = z0011001
run
# in = 0101, HEX1 = z1000000, HEX0 = z0010010
run
# in = 0110, HEX1 = z1000000, HEX0 = z0000010
run
# in = 0111, HEX1 = z1000000, HEX0 = z1111000
run
# in = 1000, HEX1 = z1000000, HEX0 = z0000000
run
# in = 1001, HEX1 = z1000000, HEX0 = z0011000
run
# in = 1010, HEX1 = z1111001, HEX0 = z1000000
run
# in = 1011, HEX1 = z1111001, HEX0 = z1111001
run
# in = 1100, HEX1 = z1111001, HEX0 = z0100100
run
# in = 1101, HEX1 = z1111001, HEX0 = z0110000
run
# in = 1110, HEX1 = z1111001, HEX0 = z0011001
VSIM 6> run
# in = 1111, HEX1 = z1111001, HEX0 = z0010010

```

Figure 5: printed tb\_part2.v input and outputs