Combinational Logic Circuits

An introduction to logic minimization techniques

Introduction

This project presents several circuit requirements that are described with higher-level, natural-worded descriptions. Your job is to create circuits that behave according to the descriptions.

When starting circuit designs from higher-level, purely behavioral descriptions like those presented below, it's a good idea to identify and follow a set of good design practices. In general, good design practices follow several steps: first, be sure you clearly understand the design intent, and exactly what is being asked; second, cast the worded description into formalisms like a block diagram and **truth table** that shows all **inputs** & **outputs** and their functional **relationships**; third, capture a **circuit** based on the formalisms (that is, write a **Verilog description**); and fourth, **implement** the circuit and **verify** its performance. Each of the requirements below presents problems that are involved enough that all of these steps should be followed.

Before you begin, you should:

- Have a working knowledge of Vivado and a functioning Boolean board;
- Be comfortable working with logic equations and logic circuits;
- Be able to describe logic circuits in Verilog;
- Know how to implement logic circuits on the Boolean board.

After you are done, you should:

- Be able to write more complex Verilog descriptions;
- Be able to analyze more complex descriptions, and design circuits to implement the behavior;
- Know how to find a minimum expression for any given logic requirement.

Background

Each of the problems in this project describes a two-state output (either on or off, true or false, 1 or 0, etc.) that is a logic functionof some number of two-state (or binary) inputs. It follows that each of these problems can be represented using a truth table, and the truth table forms the specification for a circuit. The **truth table** exactly specifies the behavior of a circuit, but not the structure. Before a physical circuit can be constructed, its structure must be defined.

For any given behavior description, any number of physical/structural circuits could be built to implement the same behavior. Of all the possible circuits that could be constructed, our goal is to find the most **efficient** one. Note that "efficiency" could imply several end goals – the most efficient circuit could use the fewest number of transistors, or it could use the least amount of power, or it could run the fastest. All of these endpoints could result in different circuits. For the most part, we will look for the circuit that uses the fewest number of transistors. Several methods have evolved to find the most efficient circuit, and they are discussed in the **theory** topics.

To solve these problems, you should cast the described behavior in a truth table, then analyze/process that truth table to find a **minimal circuit** (using the methods described in the theory documents), then capture the circuit in the **Vivado** design tool, and then **implement** and **verify** your design.

Simulation

When Verilog source files are created for more complex problems like the ones in this project, it is quite possible they contain some errors. Before implementing the design in hardware, or using it as a component in some other design, it makes sense to use a logic simulator to check the circuit's performance. The logic **simulator** creates an executable model of a circuit from a Verilog design source file, applies user-defined input signals to the circuit's input ports, and then simulates the circuit's behavior. The simulated circuit outputs are shown in a **waveform** viewer so the designer can verify that expected outputs are generated for all combinations of inputs.

There are many benefits to simulating a circuit early in the design process, before pressing on with further design work. In addition to detecting and correcting design flaws, signal timings can be checked and verified, misnamed signals can be identified, proper operations in corner cases can be examined, and so on.

For all but the simplest designs, you should adopt the habit of simulating your code and verifying your circuit's performance as soon as the source code is complete. If you start using the simulator now, with simpler circuits, you will develop some amount of proficiency. Then, as your designs become more complex, running the simulator will not feel burdensome. Like thousands of engineers before you, you will find that if you simulate and verify your code early in the design process, you will decrease your work and increase your productivity.

A background topic document presents a guided tutorial that will walk you through creating a **test bench** for the majority of five circuit in the first requirement below.

Requirements

☑ 1. Design a "majority of five" circuit

Design a majority-of-five circuit that **outputs a 1 when any three or more of its five inputs are asserted.** Implement the circuit on the Boolean board, using **5 slide switches as inputs and an LED as the output**.

The first step in the design process, understanding the objective and intent, is straightforward for this design.

The next step is to cast the requirement into an engineering formalism, which means creating a **truth table** to capture the required behavior. Then the truth table can be analyzed using **K-maps**, and a **minimal circuit** defined. In the figure below, the truth table is shown in an uncompressed "super K-map" and in an entered-variable K-map. Optimal equations can be looped from either the super map or EV map – an initial set of loops is shown. After the equations have been obtained, they can be captured in a **Verilog** design file.

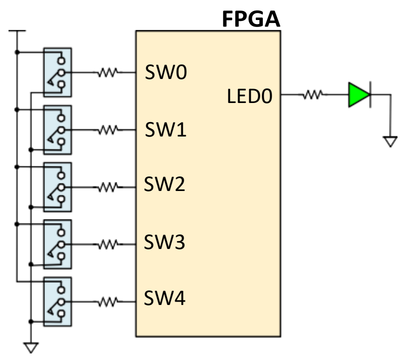
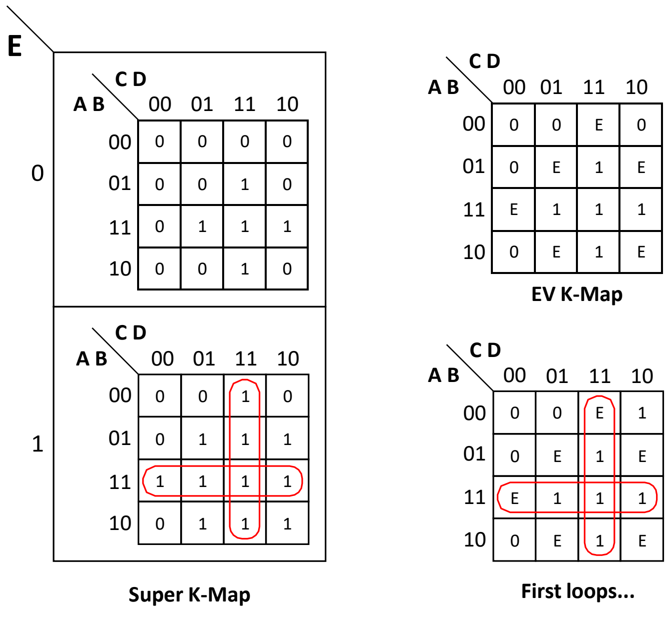


Figure 1. Majority of five block diagram

A	B	C	D	E	Y
0	0	0	0	0	0
0	0	0	0	1	0
0	0	0	1	0	0
0	0	0	1	1	0
0	0	1	0	0	0
0	0	1	0	1	0
0	0	1	1	0	0
0	0	1	1	1	1
0	1	0	0	0	0
0	1	0	0	1	0
0	1	0	1	0	0
0	1	0	1	1	1
0	1	1	0	0	0
0	1	1	0	1	1
0	1	1	1	0	1
0	1	1	1	1	1
1	0	0	0	0	0
1	0	0	0	1	0
1	0	0	1	0	0
1	0	0	1	1	1
1	0	1	0	0	0
1	0	1	0	1	1
1	0	1	1	0	1
1	0	1	1	1	1
1	1	0	0	0	0
1	1	0	0	1	1
1	1	0	1	0	1
1	1	0	1	1	1
1	1	1	0	0	1
1	1	1	0	1	1
1	1	1	1	0	1
1	1	1	1	1	1



Super K-Map, E = 0:

AB \ CD	00	01	11	10
00	0	0	0	0
01	0	0	1	0
11	0	1	1	1
10	0	0	1	0

Super K-Map, E = 1:

AB \ CD	00	01	11	10
00	0	0	1	0
01	0	1	1	1
11	1	1	1	1
10	0	1	1	1

EV K-Map:

AB \ CD	00	01	11	10
00	0	0	E	0
01	0	E	1	E
11	E	1	1	1
10	0	E	1	E

First loops...:

AB \ CD	00	01	11	10
00	0	0	E	1
01	0	E	1	E
11	E	1	1	1
10	0	E	1	E

Figure 2. Representations for the Majority of Five circuit

You might entertain an alternative way of looking at the problem, both as a way to gain more insight, and as a check on the formal design procedure using truth tables and K-maps. In fact, following a methodical design procedure is often the best way to tackle any given design, but thinking about the problem from a different angle is often advantageous as well.

In this case, with a little thinking, you might realize that all the terms in the final equation will have three logic variables and all unique three-variable terms will be included in the equation. A "5 choose 3" calculation shows there are 10 ways to choose three variables out of five; it follows that ten 3-input terms can be defined from a pool of five input variables.

You can find the logic terms needed in the final equation through looping, or perhaps you can discern them just by thinking through the combinations. Or better yet you can do both, and make sure both approaches arrive at the same solution (they do). Either way, the Verilog code below captures the requirement. Pause a moment to think about this, and make sure you agree and understand.

```
module majority_of_five(input [4:0] sw, output led);

assign led =    (sw[0] & sw[1] & sw[2]) | //ABC
                (sw[0] & sw[1] & sw[3]) | //ABD
                (sw[0] & sw[1] & sw[4]) | //ABE
                (sw[0] & sw[2] & sw[3]) | //ACD
                (sw[0] & sw[2] & sw[4]) | //ACE
                (sw[0] & sw[3] & sw[4]) | //ADE
                (sw[1] & sw[2] & sw[3]) | //BCD
                (sw[1] & sw[2] & sw[4]) | //BCE
                (sw[1] & sw[3] & sw[4]) | //BDE
                (sw[2] & sw[3] & sw[4]);  //CDE
endmodule
```

☑ 2. Design a five-way light switch

A room has 5 doors with a light switch next to each door, and one light in the center of the room. Design a circuit that allows any light switch to change the state of the light (that is, **if the light is currently off, any switch can turn it on, and vice-versa**). Use **5 switches** and **one LED** on your Boolean board to build and demonstrate your circuit.

Each of the five inputs can be on or off (a '1' or a '0'), so you can represent all possible combinations of inputs in a truth table. **The first row (all 0's) represents a state where the light is OFF.** From there, **if any input toggles to a "1", the light toggles ON. When the output is on, any new input toggle will set the light OFF again, and then any additional toggle will turn the light back ON, etc.** You must cast this behavior into a truth table, and then use that **truth table** to capture a **circuit**.

Bonus

Design a temperature indicator

A digital thermometer produces a continuously varying voltage signal between 0V and 5V, where 0V represents 0 degrees and 5V represents 100 degrees. This signal is digitized using an Analog-to-Digital converter that produces an 8-bit binary number proportional to temperature, where **00000000 represents 0 degrees and 11111111 represents 100 degrees (so each binary number represents a multiple of 100/256 degrees**).

Design a logic circuit that outputs a logic **high** signal whenever the **temperature is greater than 62.5 degrees but less than 72.5** degrees. Use **K-maps** to define the circuit, then **create** and **simulate** a Verilog description (you will need to create a test bench), and then **program** the Boolean board with your circuit. Use the **8 slide switches** to emulate the thermometer output, and **an LED** to indicate when the temperature is within the desired range.

Hint: You can make an excel spreadsheet with **0-255** in one column, scaled numbers in the next column (by multiplying the first column by **100/256**, and **rounding** to **1 digit after the decimal point**), and binary numbers in the third column (by using excel's **DEC2BIN** function on the first column of decimal numbers). Then you can identify the numbers in the **required range**, and **devise a circuit to detect only those numbers.**

Example: Binary 10110000 would convert to 68.75 degrees (68.8 degrees after rounding), which is within the desired range, and so would cause the output signal to run high.

* For Requirements 2-3, you can find example videos of tasks implemented on a Blackboard (not Boolean Board, but similar) in the following links:

https://www.youtube.com/watch?v=MdhogQrMgzo
https://www.youtube.com/watch?v=UClfsdFHt18