

Lab Worksheet 4 (Week 23)

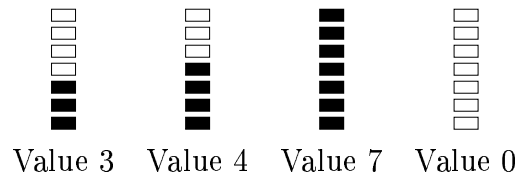
Introduction

This worksheet contains several exercises in logic design and implementation. Some of the exercises continue the tutorial work from the last tutorial before the Easter break.

Bargraph driver: design

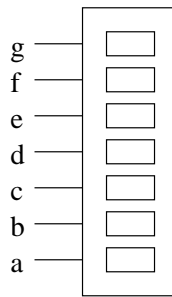
You should already have done this design work in the last tutorial. If you have not done it, or if you do not have your design with you, then you will have to do it now.

Some electronic devices present numerical information in the form of a bargraph. Typically there is a column of lights, and at any time the number of consecutive lights which are on, starting from the bottom, represents a numerical value. Music systems often use this scheme to show the overall volume setting, or the strength of the signal in a particular frequency range. For example, if the bargraph can represent a value between 0 and 7: (in this diagram, black represents a light being on, and white represents a light being off)



A *bargraph driver* is a circuit which inputs a binary number (for this exercise it will be a 3 or 4 bit number) and outputs a, b, c, \dots which are the values of the lights, from bottom to top. These values are 1 for on (black in the diagram) and 0 for off (white in the diagram).

1. Work out the truth table for a bargraph driver which inputs a 3 bit number xyz (i.e. x, y, z are the digits of a 3 bit binary number, from left to right) and outputs a, b, c, d, e, f, g (so a is the bottom light and g is the top light).
2. Work out a Karnaugh map for each output a, \dots, g .
3. From the Karnaugh maps, work out formulae for a, \dots, g . Take care to work out the simplest formulae (i.e. use the largest possible rectangles) because they will be needed later.
4. Draw a diagram of a circuit which will calculate a, \dots, g from x, y, z . When you build this circuit in LogicWorks, the outputs a, \dots, g will be connected to a component which looks like this:



so you should lay out your diagram with this in mind.

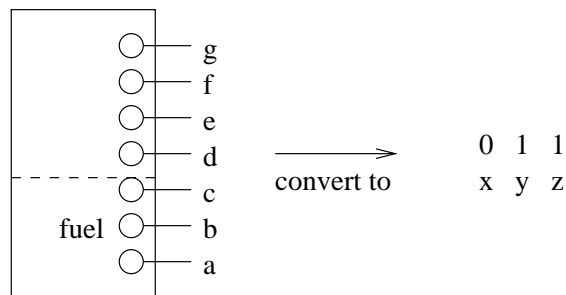
Bargraph driver: implementation

Go into AMS and set up the CS1Q exercise “Week23”

5. Start LogicWorks on the file “Exercise 5”. The file already contains a bargraph component, with inputs a, \dots, g . Build the circuit from Question 4, connecting the a, \dots, g outputs to the bargraph. Connect the x, y, z inputs to binary switches. Test your circuit by setting each 3 bit value in turn on the x, y, z switches and checking that the bargraph displays the value correctly.

Inverse of bargraph driver: design

Now consider the inverse problem: designing a circuit which inputs a, \dots, g and outputs x, y, z . For example, imagine that a, \dots, g are the outputs of a series of sensors at different heights in a fuel tank, and have value 1 if the sensor is submerged, 0 otherwise.



When the fuel is at a certain depth, a number of sensors will be submerged, from the bottom upwards. We want to convert the sensor values into a 3 bit binary value xyz which gives a measure of the fuel level.

6. One way of designing the circuit would be to think in terms of a 7 input truth table, and express x, y, z in sum of products form using minterms built from a, \dots, g . This would lead to quite complicated formulae, which would be difficult to simplify because the Karnaugh maps would be unmanageable.



























There is a simpler way which uses adders. Work out how to do this and draw a circuit diagram.

Inverse of bargraph driver: implementation

- Using LogicWorks, build and test your design from Question 6. Use a sequence of binary switches to represent the input (or, you could use your circuit from Question 5, connecting the outputs into the inputs of the circuit for this question).


Braille again

Recall the braille representation of the alphabet:

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
|  |  |  |  |  |  |  |  |  |  |
| A | B | C | D | E | F | G | H | I | J |
|  |  |  |  |  |  |  |  |  |  |
| K | L | M | N | O | P | Q | R | S | T |
|  |  |  |  |  |  | | | | |
| U | V | W | X | Y | Z | | | | |

Recall that a braille character can naturally be viewed as a 6 bit binary word, one bit for each position in the grid, in which 1 represents a bump and 0 represents no bump. For the rest of this worksheet we will think of a braille character as a 6 bit word $abcdef$ where the bits $a \dots f$ correspond to the grid like this:

$$\begin{array}{cc} a & b \\ c & d \\ e & f \end{array}$$

For example, M () corresponds to 110010 ($a = 1, b = 1, c = 0, d = 0, e = 1, f = 0$).

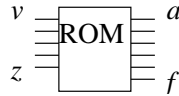
We will again be using the braille display device from Week 21, which has 6 inputs ($a \dots f$) and produces a display similar to our printed form of braille: a grid in which each position is black for a bump or white for no bump.

Suppose that we want to design a circuit which displays braille letters. To represent the 26 letters A...Z we need to use 5 bit words. A natural way to do this, which is compatible with the standard ASCII character set, is to use the words 00001...11010 (decimal 1...26). (In the ASCII character set, which uses 7 bit words, these representations are prefixed by 10 so that the letters A...Z have ASCII codes 65...90.)

If we followed our normal design process, we would work out a truth table with 5 input columns (v, w, x, y, z say) and 6 output columns ($a \dots f$ as before). This would lead to complex definitions of $a \dots f$ with no prospect of simplification because we can't easily use Karnaugh maps for more than 4 inputs.

Instead we will use a different approach, which is supported by LogicWorks and is more likely to be used in practice for complicated conversions between representations. We will

program a look-up table into a programmable read-only memory (PROM). The idea is to build a memory device which stores the entire truth table. This memory will store 32 words, each of 6 bits. It is read-only memory, so its contents can't be changed once it has been built: all we can do is look up the contents of each location. The memory has 5 inputs, corresponding to the 5 input columns of the truth table, and 6 outputs, corresponding to the 6 output columns. It looks like this:



A particular row of the truth table is specified by particular values of $v \dots z$. When used as inputs to the PROM, these values specify a particular location within the memory. The PROM will be programmed so that the contents of this location are the output columns $a \dots f$ for that row of the truth table.

In order to program this PROM in LogicWorks, we need to convert the output columns of the truth table into 2-digit hex numbers.

8. Work out a truth table with the following columns. There will be 32 rows, so use a big enough piece of paper.
 - Input columns v, w, x, y, z which run from 00000 to 11111.
 - A column showing which letter corresponds to each input combination. Remember that A is 00001. The inputs 00000, and 11011 onwards, do not correspond to letters; in these cases leave this column blank.
 - Output columns a, b, c, d, e, f which represent the pattern of bumps in the braille grid for each letter. In the cases which do not correspond to letters, put 0 in all of the output columns.
 - A column which shows the 6 bit binary number $abcdef$ as a 2 digit hex number. Hint: convert ab to the first hex digit and convert $cdef$ to the second hex digit.
9. Start LogicWorks on the file “Exercise 9”. This file already contains a braille display component. The following steps will enable you to program a PROM with the data from Question 8.
 - (a) In the “Week23” folder, open the file “BraillePROMdata” by double-clicking. Enter the 32 2-digit hex numbers from Question 8, separated by spaces or newlines (whichever you prefer). Save the file and close it.
 - (b) Back in LogicWorks (on the file “Exercise 9”), select “PROM/RAM/PLA Wizard” from the “Simulation” menu. Select “Programmable Read Only Memory” in the resulting dialogue box, and click on “Next”.
 - (c) In the next dialogue box specify 5 address lines, 6 bits per word, select “Read data from a raw hex file”, and click on “Next”.
 - (d) In the next dialogue box, click on “Select Raw Hex File”, then browse to find the file “BraillePROMdata”. You will have to select “All Files” in the “Files of type” box. When you find the file, click on “Open”. Then click on “Next”.

- (e) In the next dialogue box, enter the name “Braille” for your PROM, then click on “New Lib” to create a new component library in which to put your PROM.
- (f) In the next dialogue box, enter “Week23” as the filename for your library, and click “Save”. The previous dialogue box will return; click “Finish”.

You should now see a component called “Braille” in the component library. Select this component and place it in the design area. Connect the outputs (they are called **Out5...Out0** but they correspond to $a \dots f$) to the inputs of the braille display. Connect the inputs (called **In4...In0** and corresponding to $v \dots z$) to binary switches. Test the circuit by checking, for each combination of the switches, that the correct braille pattern is displayed.

7 segment display driver with a PROM

10. Redo the 7 segment display driver exercise using a PROM.