LAB 11 // Software
PROGRAMMABLE LOGIC ARRAYS

INSTRUCTIONS and RUBRIC

CSE 2301 – Fall 2024

INTRODUCTION

Prerequisites

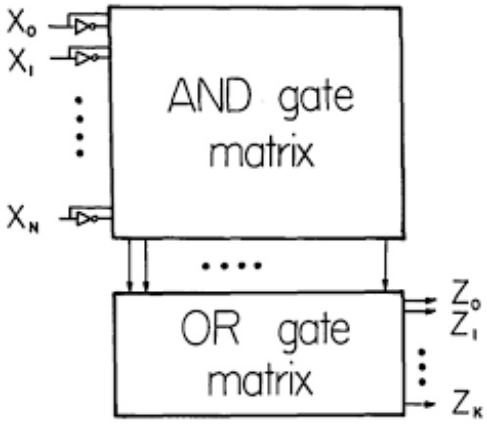
For this assignment it is expected that you are familiar with the operation of a PLA and how it is formed. Also, you need to be able to minimize a 5-variable Karnaugh Map.

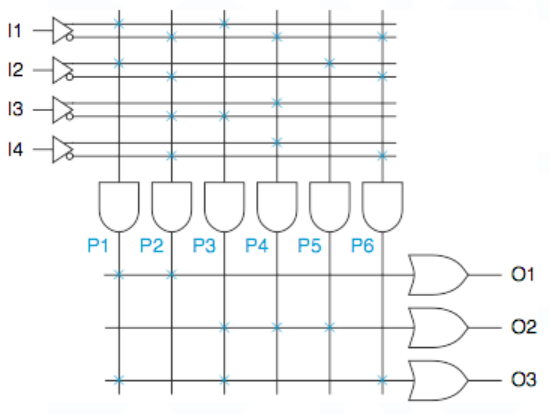
Objectives

The objectives of this assignment are to reinforce the idea of PLAs as function generators and to show how they are constructed at the gate level and implemented in a software PLA package using LogicWorks.

Background

There are many different ways to implement logic functions and today's exercise is to focus on one of the Programmable Logic Devices (generic term) known as a PLA. This is similar to a PAL but not identical to it. A PLA is a field programmable device, which means that you can quickly make connections within the device to create a set of logic functions to suit your output requirements. The 'field programmable' description is as opposed to a 'mask programmed' PLD in which the connections have been made during the manufacturing process.





PART 1:

Manual Output

The outputs of the circuits to be designed using a gate-level representation have to follow the truth tables below. In this case there are five inputs (C2, C1, C0, A, B) and one output (F).

C2	C1	Co	A	B	F
0	0	0	0	0	1
0	0	0	0	1	1
0	0	0	1	0	1
0	0	0	1	1	1
0	0	1	0	0	0
0	0	1	0	1	1
0	0	1	1	0	1
0	0	1	1	1	1

C2	C1	Co	A	B	F
1	0	0	0	0	1
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	0
1	0	1	1	0	0
1	0	1	1	1	1

C2	C1	Co	A	B	F
0	1	0	0	0	1
0	1	0	0	1	1
0	1	0	1	0	1
0	1	0	1	1	0
0	1	1	0	0	0
0	1	1	0	1	1
0	1	1	1	0	1
0	1	1	1	1	0

C2	C1	Co	A	B	F
1	1	0	0	0	1
1	1	0	0	1	0
1	1	0	1	0	0
1	1	0	1	1	0
1	1	1	0	0	0
1	1	1	0	1	0
1	1	1	1	0	0
1	1	1	1	1	0

YOUR TASK – Populate a 5-variable K-map with the data above (same data set, split up for viewability). There will be no "don't care" states. Minimize the function F (you will be graded on your minimization) and complete a LogicWorks circuit using AND and OR gates that satisfies the minimized function. Test the circuit exhaustively and then present to a TA.

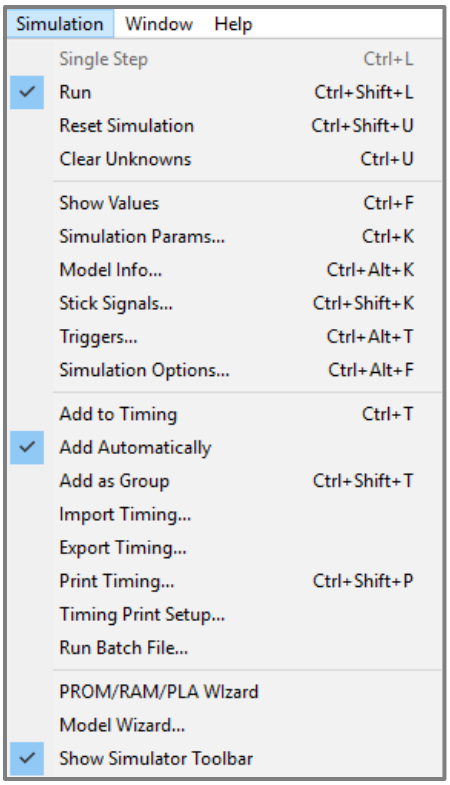
PART 2:

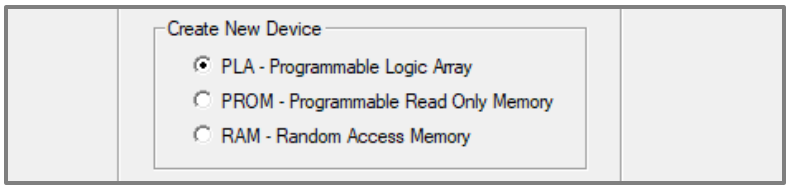
Building a PLA

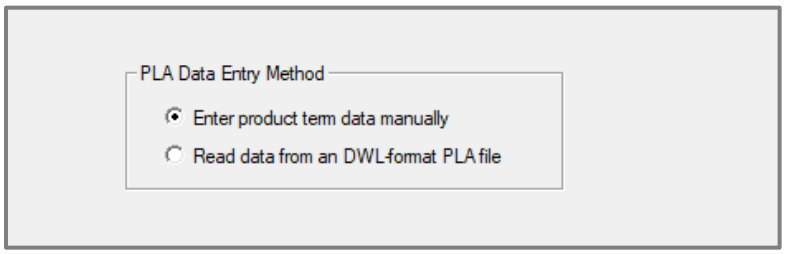
Developing large K-maps for tables like the one in part 1 is tedious, but there is a more agreeable solution in the form of PLAs. Instead of minimizing equations, we need only program the desired minterms into the PLA, and the device takes care of the rest. In a PLA, every AND gate overlaps with every input and its inversion, and every OR gate overlaps with every AND output, so we can express any desired equations.

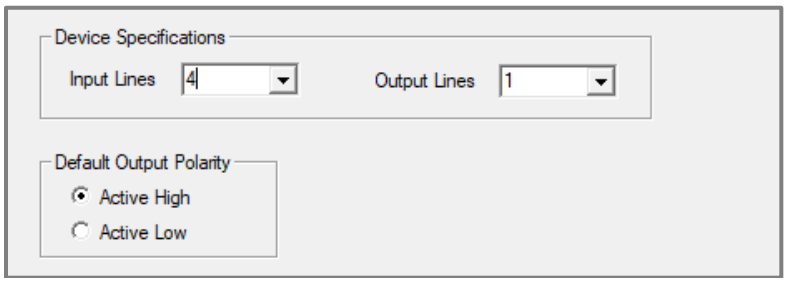
Let's take a look at how to build a PLA symbol in LogicWorks. We'll work with a simple example shown on the next page.

1. Start by navigating to Simulation > PROM/RAM/PLA Wizard.
2. Select "PLA", then "Enter product term data manually".
3. For device specifications, we'll have 4 input lines and 1 output, but if you had more than one equation, you could define more outputs. Default polarity should be "active high".



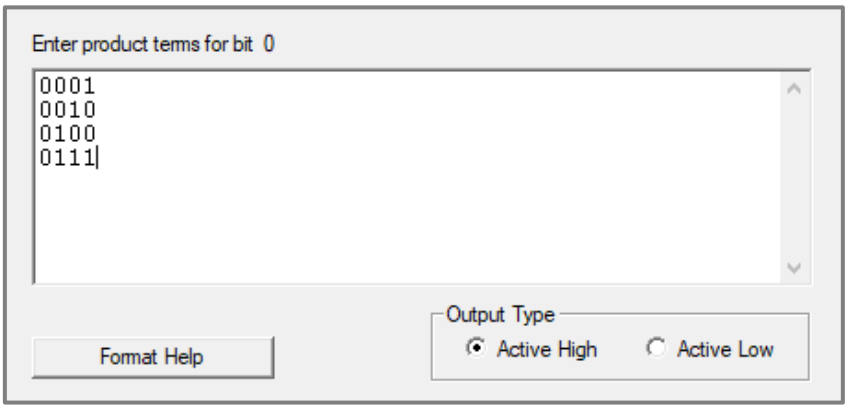






4. After selecting next again, you'll be brought to a screen that says, "enter product terms for bit `0`". This is referring to the first output (recall that we only defined one for this symbol). In this box, we will enter all of the minterms (DCBA values where F = 1) separated by a newline. Because our output type is active high, whenever `0001`, `0010`, `0100`, or `0111` are detected, output bit `0` will be driven high.

D	C	B	A	F
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1

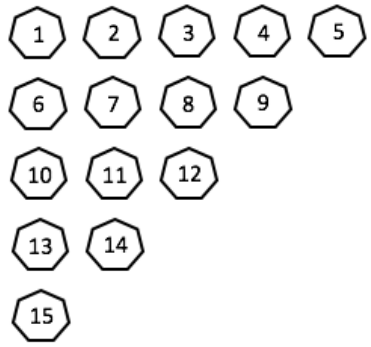


5. Select next, then name your symbol and save it to a library.
6. You can now place the symbol in a blank LogicWorks document to check that it works. This is an exercise. You do not need to demo.

PART 3:

Triangular Parity Code

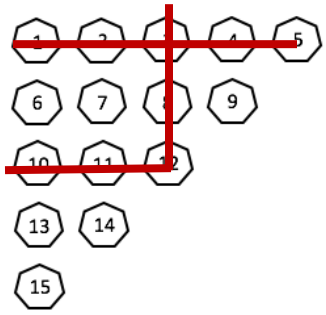
This section ties together the topic of error detection/correction with the use of a PLA to find a single error and determine its location. Remember the triangular code used in transmission of data? Here is a small example:



Check A. Position 5 checks parity for bits 1,2,3,4,5
Check B. Position 9 checks parity for bits 4,6,7,8,9
Check C. Position 12 checks parity for bits 10,11,3,8,12
Check D. Position 14 checks parity for 13,2,7,11,14
Check E. Position 15 checks parity for 1,6,10,13,15

Note that each numbered bit appears in two parity checks except for the parity bits in positions 5,9,12,14, and 15. For example bit 2 is used in parity checks A and D, bit 8 is used in parity checks B and C.

We will EVEN parity. Since there are 15 bit positions to check for an error (and we must have a "no error" condition) there are 16 overall checks. We can understand, graphically, how we might check for the location of an error, but converting that geometric result into logic is less clear…



Say that we had an error in bit position 3.
Parity bits ABCDE check for even parity in their lanes, and both A and C return an error. The common bit between these checks A [1,2,3,4,5] and C [10,11,3,8,12] is 3, so 3 is in error.

However, weighting ABCDE to indicate the position of that error logically does not appear to be possible, so we'll need a new scheme to accommodate it.

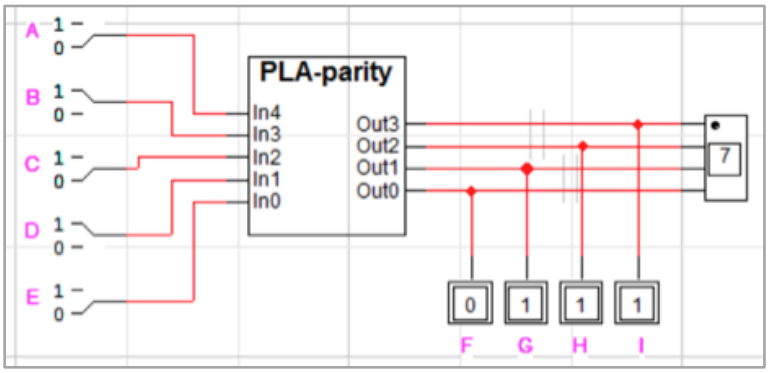
The plan is to use four more bits named FGHI and create a truth table representing the bit in error based on the check values of ABCDE. The table has been started for you. It will have 16 rows total (which should cover all possible FGHI values). Note also that if only one parity bit is in error, the parity bit itself is in error.

Inputs					Outputs				
A	B	C	D	E	F	G	H	I	Error
0	0	0	0	0	0	0	0	0	None
1	0	0	0	0	0	1	0	1	5
1	1	0	0	0	0	1	0	0	4
1	0	1	0	0	0	0	1	1	3

…and so on…

YOUR TASK – Design a PLA to give you the FGHI outputs (from ABCDE) and therefore the position of a theoretical error. You will have to start by populating the entire table, but

recognize that we are only tracking those states in which no parity bits are high (no error), one parity bit is high (parity bit in error), or two parity bits are high (data bit in error). The PLA will output zero for any non-programmed inputs. Your final product should be a circuit much this one. Do *not* create a full parity generator and checker, as you did in lab 6.



SCORING

There is no theory for this assignment, but some of the questions will cover content on the final, which is cumulative.

Demo Requirements [5 pts]:

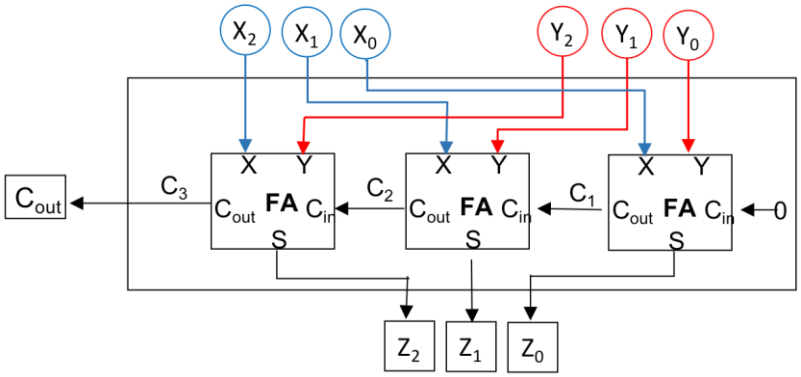
- ✓ Show TAs the LogicWorks implementation of your 5-variable Karnaugh map circuit from Part 1 **(2.5 pts).**
- ✓ Show TAs your PLA parity generator **(2.5 pts).**

Report Requirements [2 pts]:

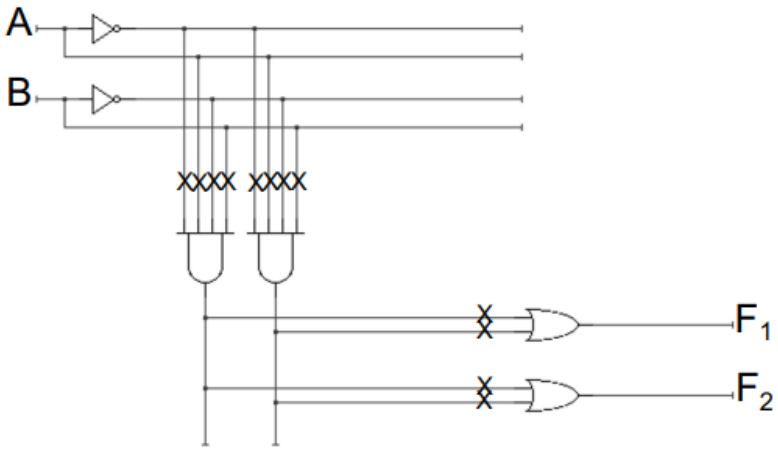
- ✓ [1.4] Deliverables:
  - [0.7] Include your 5-variable Karnaugh map from part 1. Points will be deducted if the optimal solution is not provided.
  - [0.7] Present your complete truth table from part 3. It will be graded for accuracy.
- ✓ [0.6] Discussion section. Should conform to standard lab report guidelines.

Practice Questions [3 pts]:

- ✓ [1.5] Question 1:
  A circuit for adding two 3-bit 2's complement numbers (X2X1Xo and Y2Y1Yo) that uses Full Adder (FA) components is shown below. Write the full logic expression to detect overflow.



- ✓ [1.5] *Question 2:*
  Draw ***one*** PLA diagram for the following equations:
  $F = C'A+BC$
  $G = CB'A' + BC$



Example PLA diagram.