LAB 2: COMBINATIONAL CIRCUITS (PART ONE)

This lab will focus on exploring the uses of various key combinational circuit components like encoders and decoders, comparators, etc., for a special use case: Tita Jo's seaside eatery.

As in Lab 1, please use Logisim-evolution or CircuitVerse when instructed to implement or design a circuit.

Part 1: Reading Orders

Tita Jo has 8 items on her eatery's menu: Sinigang, Lumpia, Bulalo, Manok Inasal, Lechon Kawali, Kanin, Taho, and Halo-Halo.

With lots of customers (and most of the waiters unavailable for most of the weekdays when things get busy), you will have to design a Customer Console system, with an "Ordering Machine" around three components:

Customers will get a *bubble sheet*, which they put into a *bubble card reader*. The bubble card readers can be considered 3-output machines, which return an output 0 for the corresponding output rail if the bubble is unfilled, and 1 if it does. *However, the bubble card reader can only read up to three bubbles at a time.*

These bubble sheet readers feed their outputs to an "Order Machine", which returns 8 possible outputs, connected to 8 different LEDs in the kitchen, assigned to turn on when an order is made.

The local electronics shop has completely run out of 3-bit decoders. As such, you will only have access to 2-bit decoders, AND, OR, and NOT gates. **With these resources, design and implement an "Ordering Machine."**

SOLUTION:

There are 8 potential dishes that can be ordered, and as such, each dish can be associated with a three-bit input. For half of these inputs, the MSB is 1 and for the other, the MSB is 0. It's sufficient to therefore split the dishes into two groups based on this distinction, and

then distinguish between dishes in each group by associating them with one of the outputs of a 2-bit decoder.

For our purposes going forward, we're going to map the dishes to three-bit inputs X ($x_0x_1x_2$) as follows:

Dish	X
Sinigang	000
Lumpia	001
Bulalo	010
Manok Inasal	011
Lechon Kawali	100
Kanin	101
Taho	110
Halo-Halo	111

END OF SOLUTION

Part 2: Accepting by Availability

Availability of ingredients can vary greatly in the eatery from day to day, and *it is very important to ensure orders for unavailable dishes are not accepted*. In other words, if a customer orders a dish, but the ingredients are not available, we need the corresponding light indicator in the kitchen *not to turn on*. As such, the "Order Machines" implemented in Part 1 must receive an "Availability Mechanism" upgrade.

There are 8 ingredients in the pantry, and Tita Jo has provided you with her cookbook. Every ingredient is necessary, but some can be substituted for others:

Dish	Ingredients
Sinigang	Rice Pork or Vegetables
Lumpia	Pork or Vegetables
Bulalo	Beef or Chicken Rice or Vegetables
Manok Inasal	Chicken Rice Vegetables

Lechon Kawali	Pork Rice Vegetables
Kanin	Rice Tofu Vegetables
Taho	Sugar Tofu
Halo-Halo	Sugar Ice Cream

Inventory is taken every day, and input rails are provided to r each ingredient, to indicate when it is available or not.

Your local electronics shop, this time, has a backlog. Therefore, for this task, you will only have access to NOR and NOT gates. **Implement the "Availability Mechanism" upgrade to the Customer Console.**

SOLUTION:

Looking at the list of ingredients provided by the cookbook, there are clearly 8 unique ingredients, so our 8-bit ingredient input, I ($I_0I_1I_2I_3I_4I_5I_6I_7$) will have each bit associated with one of the ingredients.

At the end of the day, we will need to produce an 8-bit availability output A ($A_0A_1A_2A_3A_4A_5A_6A_7$), so that each output A_i can be used as an input to an AND gate, the output of which will go to the kitchen to ping them that an order has been accepted.

It's worth noting that while in principle, each A_i is dependent on the full 8-bit input I, it is actually only dependent on a subset of the 8-bit. (i.e., if an A_j is not explicitly dependent on I_k, then it is treated as a don't-care input for A_j) It also helps that once each bit from I is associated with each ingredient,

Ingredient	Rice	Pork	Chicken	Vegetables	Sugar	Ice Cream	Beef	Tofu
I_j	I_0	I_1	I_2	I_3	I_4	I_5	I_6	I_7

then Boolean expressions for A become:

A_0	$I_0(I_1 + I_2)$
A_1	$I_1 + I_3$
A_2	$(I_0 + I_3)(I_2 + I_6)$

A_3	$I_0 I_2 I_3$
A_4	$I_0 I_1$
A_5	$I_0 I_3 I_7$
A_6	$I_4 I_7$
A_7	$I_4 I_5$

END OF SOLUTION

Part 3: Payment

The last interaction left to automate is the customer payment. Each of the dishes cost an integer multiple of ₱5. (e.g., an order of Lumpia costs ₱5, an order of Lechon Kawali costs ₱10, and so on.) **Assign a price to each dish, and make sure each price is a unique multiple of ₱5, up to ₱40 at most.**

Tita Jo needs a new Payment Mechanism to be implemented into the Customer Console, as an additional check before an order can be accepted, in the same way the Availability Mechanism was a check before an order can be accepted (i.e., for the appropriate LED in the kitchen to switch on).

When paying, customers deposit their coins into a coin machine, where only ₱5 coins are accepted. This coin machine has a useful feature: it provides an 8-bit output C ($C_1, C_2, \ldots, C_8$), such that the jth bit (and all less significant bits) are turned "on" once j many coins are deposited. In particular, if the Customer pays j many ₱5 coins (totaling a payment of ₱(5*j)), then C_m =1 for $m \leq j$, while C_k = 0 for k > j.

Implement the "Payment Mechanism" update to the Customer Console. This time, the electronics shop will finally give you access to all of the components within. (In practical terms, this only includes components discussed in class up to the

assignment of this lab.) However, money is also tight; the business cannot afford more than three additional components in this design.

SOLUTION:

The key here is that we want to map all possible payment inputs C to binary representations of the number of coins deposited, as well as to map each dish to a number of coins required as payment, and to find a way to compare the two numbers.

For the latter, we can simply take X from the table in Part 1 to be the "cost" of said dish. In particular, if the real cost of the dish would be Y coins, then X is the binary representation not of Y, but Y-1. (In the end, this will not produce an issue but it's still worth noting.)

For the former matter, we will need a 3-bit priority encoder, which will take an 8-bit input; if we feed the jth bit from C to the jth input bit for the encoder, then an input C of 00000001, which represents a payment of 1 coin, will return a value of "000" (the binary representation of 0, which is 1 less than 1, the number of coins paid). An input C of 00000011 will be associated with a payment of 2 coins and will return a value of "001" (again, the binary representation of 1, which is 1 less than 2, the number of coins paid), all the way to an input C of 11111111 being associated with a payment of 8 coins (totaling to the maximum 40 pesos you can charge) returning a value of "111" (which is the binary representation of 7, which is 1 less than 8, the number of coins paid; 10 points to Gryffindor if you can guess the connection to the previous paragraph and why I noted the relationship between X and Y).

To accept a payment, the result of the former must be greater than or equal to the latter. To make this comparison, we simply have to use the two as inputs to a 3-bit comparator, and then, take the OR of the ">" and "=" outputs of the comparator, and then use the output of this OR gate as the last input of the AND gates connecting to the LEDs in the kitchen. This ensures that so long as the payment is greater than or equal to the cost, the payment, and thus the order, is accepted. (Depending on which value is A and B in the comparator, you might have to take the OR of "<" and "=", but this can just be resolved by just swapping the inputs to the comparator, so you can follow the original plan above.)

END OF SOLUTION