LAB 3: COMBINATIONAL CIRCUITS (PART TWO)

This lab will continue exploring the uses and functions of various key combinational circuit components like encoders and decoders, multiplexers (MUX), and comparators. This time, it will be done in the context of tackling, improving, and upgrading a pre-existing system: a Trigger System for a particle accelerator.

A lot of particle accelerators produce a lot of collisions, decays, and other events, but not all computer systems will be able to store all of the data generated. A Trigger System is therefore necessary to determine whether the event is "interesting" enough for its data to be kept for further analysis.

This particular particle accelerator is having a rough time of it, due to a faulty, flawed, and outdated Trigger System. You are now tasked with debugging and upgrading it to suit the research group's new needs.

(Logisim-evolution will be used in this lab to load the pre-designed circuits, and to implement upgrades.)

Part 1: Momentum Selector

One of the most important observables is the momentum of the incoming particles, **s**. The detector looks at 8 levels of momentum:

- Level 0, when $0 \leq s \leq 0.5$ TeV,
- Level 1, when 0.5 TeV $< s \leq 1.0$ TeV,
- Level 2, when 1.0 TeV $< s \leq 1.5$ TeV,
- and Level 3, when 1.5 TeV $< s \leq 2.0$ TeV, etc.

Correspondingly, every event produces a 4-bit output S from the detector, propagated into the Trigger, with S_0 as the LSB, and S_3 as the MSB, obeying the rule that, if an event achieves momentum Level j, then $S_j = 1$ for j = 0,...,j, and S_k = 0 for any k > j. (E.g., if an event achieves momentum Level 2, then $S_0 = S_1 = S_2 = 1$, but $S_3 = 0$.)

The Momentum Selector takes two 8-bit inputs: the S input provided by the detector, and a 8-bit T input provided by the user. The T-input represents the threshold set by the user, with each T_L associated with Level L. If a user wants the Trigger to only accept events of Level L or above, then T_L is set to 1, while all other T_M are set to 0. The final output of the Momentum Selector must be 1 when an event is accepted (i.e., when the event momentum s is above the threshold set by the user), and 0 otherwise.

The present system is built only for 4 Levels of momentum s. Now, the research group wishes for better resolution, and wants the system upgraded for 8 Levels of momentum s. The group has already upgraded the detector and user interface to now provide 8-bits for inputs S and T.

However, the group budget will not provide for any additional comparators, or comparators with higher bits. You will, however, have access to NOT, AND, OR, and XOR gates. ***Upgrade the Momentum Selector for 8 Levels of momentum s, so that the output "Trigger" is 1 if and only if the momentum s is greater than or equal to the threshold provided by the user.***

SOLUTION:

There are different ways to go about this, but the first step is to map the possible inputs to binary inputs we can use for the two-bit comparators. This can easily be done with the Priority Encoder (notice that when S is "Level 0", i.e., S = 00000001, then the output of the encoder is "000", and when S is "Level 4," i.e., S = 00001111 then the output of the priority encoder is "010").

This naturally leads to the comparator being the natural component to use to determine whether the momentum associated with S can clear the threshold set by T, as you simply feed S and T each into a 3-bit priority encoder to get outputs Sb and Tb, then use them as inputs to a comparator to get the three outputs ">", "=", and "<". The output, "Trigger", will have to be the OR of the ">" and "=" outputs of this 3-bit comparator.

With only a two-bit comparator, we will have to improvise a little. There is some variation in how to do this, but one way to do it is to use the two-bit comparator to compare the two most significant bits of Sb and Tb. And then, we can design a 1-bit comparator from scratch to compare the LSBs of Sb and Tb. The key is that the 2-bit comparator's output takes priority, in that if it returns “>” or “=” as 1, then “Trigger” must be 1, no matter the result of the 1-bit comparator. Similarly, if the 2-bit comparator returns “<” as 1, then “Trigger” must return 0, no matter the result of the 1-bit comparator.

It is only when 2-bit comparator’s result is “=”, will the 1-bit comparator decide the result of the S vs. T comparison.

And when designing the 1-bit comparator, one important thing to check is whether the two bits are the same, which can be done with an XOR gate. If it returns 1, then the result of the 1-bit comparator is “=” returning 1, and the other outputs 0. If Sb’s last bit is 1 and Tb’s last bit is 0, then the 1-bit comparator returns “>” as 1 and the other outputs 0, and conversely if Tb’s last bit is 1 and Sb’s last bit is 0, then the 1-bit comparator returns “<” as 1 and the other outputs 0.

With all of these principles in mind, one can get the result <ECE_Project _3_MomentumSelectorSolution> in the Github.

END OF SOLUTION

Part 2: Particle Flagger

The particle accelerator’s detector is built to detect all 6 kinds of quarks: up, down, charm, strange, top, and bottom. As such, it also provides a 6-bit input into the Trigger system, Q, where Q_0 is the LSB and Q_5 is the MSB. Each Q_j = 1, when the quark associated with it is detected, and 0 if not. The quarks are each assigned to the Q inputs as follows (although you can reassign them, **but if you do so, state this clearly and explicitly):**

Quark	**Q input bit**
up	Q_0
down	Q_1
charm	Q_2
strange	Q_3
top	Q_4
bottom	Q_5

Table 1: Assignments of quarks to Q 6-bit inputs

The user will also have access to another interface, allowing them to select for events to be accepted only when the chosen particles are detected. The user interface thus provides a 6-bit input C, where C_j = 1 if the user wants events with the associated quark detected. Each C_j bit is associated with the same quark as each Q_j bit. (E.g., Q_3 is associated with the strange quark, and as such C_3 is also associated with the strange quark)

Design a Particle Flagger upgrade to the Trigger System, so that the output "Trigger" returns 1 if and only if the particles selected by the user are detected, in addition to the condition set in Part 1. *The budget is even tighter this go around, and you are limited to 6 components, only one of which can be an AND gate, and any AND/OR gates available can only take two inputs. Every component above the limit, or use of gates with more than one input will result in a point reduction of 2.*

SOLUTION:

Based on the information, the output "Trigger" needs to be 1 if and only if the inputs Q and C are the same.

One simple, brute-force way to do this check is by simply taking the XNOR of each bit, and if the output of any one of these gates is 0, then the output should be 0, so we would simply take the AND of all the results of these XNOR gates. But, with a limited number of components that we can use, we will have to be more clever.

We can instead make use of decoders and multiplexers. Multiplexers are naturally useful to us here, since in a way, they are meant to decide whether data should be accepted; in particular, they can take the outputs of decoder as the data inputs, and a selector input, and will only return 1 if the output of the decoder which serves as the data input, had an input the same as the selector input. Then, "Trigger" should simply be the output of the MUX.

It is possible to use this method compare the first three bits and the last three bits of Q and C each. Then, "Trigger" should only return 1 when both MUXs for each comparison give the all clear (i.e., both return 1), so the "Trigger" should be the AND of the outputs of each MUX—the only AND gate, which only needs two inputs, as in <ECE_Project_3_ParticleFlaggerSolution> in the Github repository.

END OF SOLUTION