**Design and Simulation of a Muller-C Element**

Elikem Asudo Tsatsu Gale-Zoyiku

Pascal Okoli Mathias

Joseph Narteh Lartey

Shinell Naa Ayeley Aryee

Computer Science and Information Systems Department

[22-23\_SEM2\_CS221\_A] - Discrete Structures and Theory

Ayawoa Sitsope Dagbovie (Dr.)

April 19, 2023

## Introduction

The Muller-C element is a standard logic gate commonly used to synchronize independent processes in most asynchronous electronic circuits. It outputs 0 when all inputs are 0, it outputs 1 when all inputs are 1, and it retains its output state otherwise. In this report, we will design and simulate a two-input Muller C-element using Logisim. We will explain the theories and concepts behind the Muller C-element, such as its applications in asynchronous systems. We will also describe the methodology we followed to design and simulate the circuit using Logisim, such as the choice of components, the wiring of connections, and the testing of functionality. The rest of the report is organized as follows: Section 2 provides further background information on the Muller C-element. Section 3 presents the design and simulation of the circuit using Logisim. It also discusses the results and analysis of the simulation. Section 4 concludes the report.

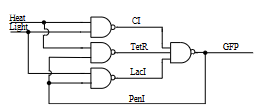
## Background

A Muller-C element is a type of logic gate that is widely used in asynchronous circuits and systems. It is also known as a C-gate, a hysteresis flip-flop, a coincident flip-flop, or a two-hand safety circuit. It has the property that it outputs 0 when all inputs are 0, 1 when all inputs are 1, and retains its output state otherwise. This makes it helpful in synchronising independent processes that do not have a common clock signal. A Muller-C element can be implemented using various electronic components, such as transistors, resistors, capacitors, or logic lookup tables (LUTs) in field-programmable gate arrays (FPGAs). It was first introduced by David E. Muller in 1955 and used in the ILLIAC II computer. Some real-life applications of Muller-C elements include:

* Designing genetic circuits that can change the behaviour of organisms in useful ways, such as producing drugs, cleaning up toxins, or detecting and killing tumour cells. A genetic Muller-C element can be constructed using synthetic DNA strands that act as inputs and outputs and enzymes that act as logic gates. A genetic Muller-C element can help coordinate the actions of different genes and proteins in a cell without relying on external signals.
* Designing asynchronous processors that can operate faster and more efficiently than synchronous ones, as they do not need to wait for a clock cycle to complete an operation. A Muller-C element can create pipelines, feedback loops, and handshake protocols that allow different processor parts to communicate and cooperate without timing conflicts.
* Designing fault-tolerant systems that can recover from errors and continue functioning normally. A Muller-C element can detect and correct glitches or transient faults that may occur due to noise, interference, or physical damage. A Muller-C element can also be used to implement self-timed circuits that can adapt to changes in environmental conditions, such as temperature or voltage.

A Muller-C element can be implemented using different techniques, such as majority gates, sum-of-product circuits, Schmitt triggers, or genetic circuits. A genetic Muller-C element is a synthetic biological circuit that mimics the behavior of a Muller-C element using genetic components such as promoters, repressors, and inducers.

A practical implementation of a genetic Muller-C element can be seen in the diagram below. It is implemented in a genetic circuit for the expression of green fluorescent protein (GFP). The inputs are heat and light, and PenI, an enzyme activated by the expression of GFP so it is effectively GFP itself. GFP is produced when both heat and light are present. Given that GFP is being produced, the absence of either light or heat does not change the state of the circuit. The only way GFP will seize to be produced is if both heat and light are absent.



## Methodology

The purpose of this section is to provide a detailed account of the steps we took to design and simulate a Muller-C element. The Muller-C element is a logic gate that can be implemented in various ways. We aimed to better understand its workings by selecting a two-input Muller-C implementation to analyse. This methodology section will outline the research conducted, the steps taken to draw truth tables, the circuit design using Logisim, and the testing of the circuit values with different input combinations.

Research on the various implementations of the Muller-C element

The first step in our methodology was to research the various implementations of the Muller-C element. We reviewed high-level sources on the implementation of the Muller-C element. We identified a paper describing the implementation of a genetic Muller-C element which described how proteins could be expressed more efficiently if the reaction pathway was modeled after a Muller-C element, such that the expression of the protein would be a factor in the further expression of the said protein (Nguyen et al.). We also consulted other sources, such as (source needed), which explained how to implement a Muller-C element on a LUT-based FPGA. These papers provided valuable insight into the Muller-C element's workings, and we used it as a primary reference for our analysis.

Selection of a two-input Muller-C implementation to analyse

|  |  |  |
| --- | --- | --- |
| **Xn** | **Yn** | **Zn** |
| **1** | **1** | **1** |
| **1** | **0** | **Zn-1** |
| **0** | **1** | **Zn-1** |
| **0** | **0** | **0** |

After reviewing the paper and understanding the Muller-C element's workings, we chose a two-input Muller-C element because it is the simplest case. The idea remains the same as inputs increase. We drew a truth table for the Muller-C element based on its definition, as shown in Table 1. The subscript n represents a current input or output state, while a subscript n-1 means the previous state.

Table 1

Drawing Truth Tables

|  |  |  |
| --- | --- | --- |
| **Xn** | **Yn** | **Zn** |
| **0** | **0** | **0** |
| **0** | **1** | **0** |
| **1** | **0** | **0** |
| **1** | **1** | **1** |

|  |  |  |
| --- | --- | --- |
| **Xn** | **Yn** | **Zn** |
| **1** | **1** | **1** |
| **1** | **0** | **1** |
| **0** | **1** | **1** |
| **0** | **0** | **0** |

To gain further insight into the Muller-C element's behaviour, we drew truth tables for the Muller-C element as if we started from both inputs being 0 and then as if both inputs started from 1. This gave us two truth tables that helped us understand how the Muller-C element works (see Tables 2 and 3).

Table 2 Table 3

Design of the circuit for the Muller-C element using Logisim

We then looked at two possible output expressions of a typical two-input Muller-C element, which are

**Zn = (Xn . Zn-1) + (Yn . (Xn + Zn-1),**

and

**Zn = ( Xn . Yn ) + Zn-1 . ( Xn + Yn ).**

We drew the circuit for the first expression using Logisim, as shown in Figure 1. We chose this implementation because the action of the previous state of the circuit is very evident.

Diagram, schematic

Description automatically generated

Figure 1: Circuit of two-input Muller-C element

Simulation and testing of the circuit values with different combinations for Xn and Yn

We then tested the circuit values with different combinations for Xn and Yn to see if we would get the same values we observed in our truth tables. We initially got an oscillation error because our AND and OR gates were not in the correct order. We rectified this (using the output expression as a guide) and again tested the circuit. We got the expected values this time, as shown in Figures 2 - 8.

Diagram, schematic

Description automatically generated

Figure 2: Testing our circuit with both inputs set to 0. The output state of the circuit is 0.

Diagram, schematic

Description automatically generated

Figure 3: Testing our circuit by setting X to 1 and maintaining Y at 0. The output state (Z) of the circuit remains 0.

Diagram, schematic

Description automatically generated

Figure 4: Testing our circuit by setting Y to 1 and X to 0. The output state of the circuit (Z) remains 0.

Diagram, schematic

Description automatically generated

Figure 5: Testing our circuit by setting X and Y to 1. The output state of the circuit (Z) is now 1.

Diagram, schematic

Description automatically generated

Figure 6: Testing our circuit by setting X back to 0 and leaving Y at 1. The output state of the circuit (Z) remains 1 even though one of the inputs has been turned off. Also, this is the opposite of what happened in Figure 4, even though the input states are the same.

Diagram, schematic

Description automatically generated

Figure 7: Testing our circuit by setting X back to 1 and Y to 0. The output state of the circuit (Z) remains 1 even though one of the inputs has been turned off. Also, this is the opposite of what happened in Figure 3, even though the input states are the same.

Diagram, schematic

Description automatically generated

Figure 8: Testing our circuit by setting X and Y to 0. The output state of the circuit (Z) is now 0.

Figures 2 to 8 follow the pattern from our truth tables: when the circuit starts from X = 0 and Y = 0, then any time X ≠ Y, the output of the circuit, Z = 0, until X = Y = 1. Also, when the circuit starts from X = Y = 1, X ≠ Y, Z = 1 until X = Y = 0.

Conclusion

In conclusion, the methodology section outlines the various steps to design and simulate a Muller-C element. Our methodology involved researching the Muller-C element, drawing truth tables, designing the circuit using Logisim, and testing and rectifying the circuit values. We gained a deeper understanding of the Muller-C element's workings through these steps and successfully designed and simulated a two-input Muller-C element.