Comparing the Theoretical Limit of Linear Substring Search to Practical Implementation

Aiden Swayne

September 14, 2024

1 Introduction

This project is a subsequence identifier implemented using only MOSFET transistors, specifically designed to determine whether a given substring is contained within another string and where the substring is located if it is present. This circuit was designed to run independently of its data source's speed, which is why so much of the circuit is asynchronous. The asynchronous portions of the circuit allow the data source to run at its maximum speed when loading in data and the circuit to run at its maximum speed when searching for the subsequence. The circuit was also designed for expandability. By increasing the width of wires that carry the lengths of sequences and expanding the data storage blocks, the maximum sequence and subsequence lengths could increase considerably.

2 Background on Logisim

Logisim is a purely logical simulation for circuit diagrams and VHDL code. It carries values across wires instantly and does not have any simulations of voltage. This project was not intended to have applications, so using Logisim to test it is viable. Logisim wires can have four values: 1, 0, Undefined, and Error. Note that 0 is not the default value for a wire. Wires have a default value of Undefined and must be actively pulled to 0 by one of its input blocks. To bring a wire to 1 or 0, at least one of its input blocks must output that value. Wires with a value of 1 are light green, and wires with 0 are dark green. The Undefined value is overridden by any other value, so it arises only when all of the blocks a wire connects to output the value Undefined or when the wire has no inputs. Wires carrying Undefined are colored blue. Though it is present in every basic logic gate, the sections on multiplexers and the SRAM use this property more explicitly than most. Error arises when conflicting 1 and 0 values are simultaneously placed on a single wire by different blocks and when at least one block outputs Error onto the wire. Wires carrying Error are red. Wires carrying 4 or 8 bits at once are colored in black, regardless of their value. Unfortunately, 4 and 8-bit wires are indistinguishable, and the number of bits a black wire carries must be determined by context. The purely logical nature of Logisim allows for some nonsensical constructions that have been avoided in this circuit. For example, a transistor with its inputs and outputs connected can carry a value of 1 indefinitely as long as its gate is never closed. Logisim simulates in steps, advancing across a wire or through a pin or transistor every step. This stepping behavior gives it some rudimentary propagation delay, though all delays are equal in length. These delays are most present in the basic logic gates, which make heavy use of pointless transistors whose gates are always open to introduce delays and synchronize the changing of logic values across Logisim iterations.

3 Main Circuit Explanation

The circuit consists of 3 general regions: One for retrieving subsequence data, enclosed in red; a similar region for retrieving main sequence data, enclosed in blue; and a comparison/output region to compare the values and compute the outputs, which is the rest of the circuit.

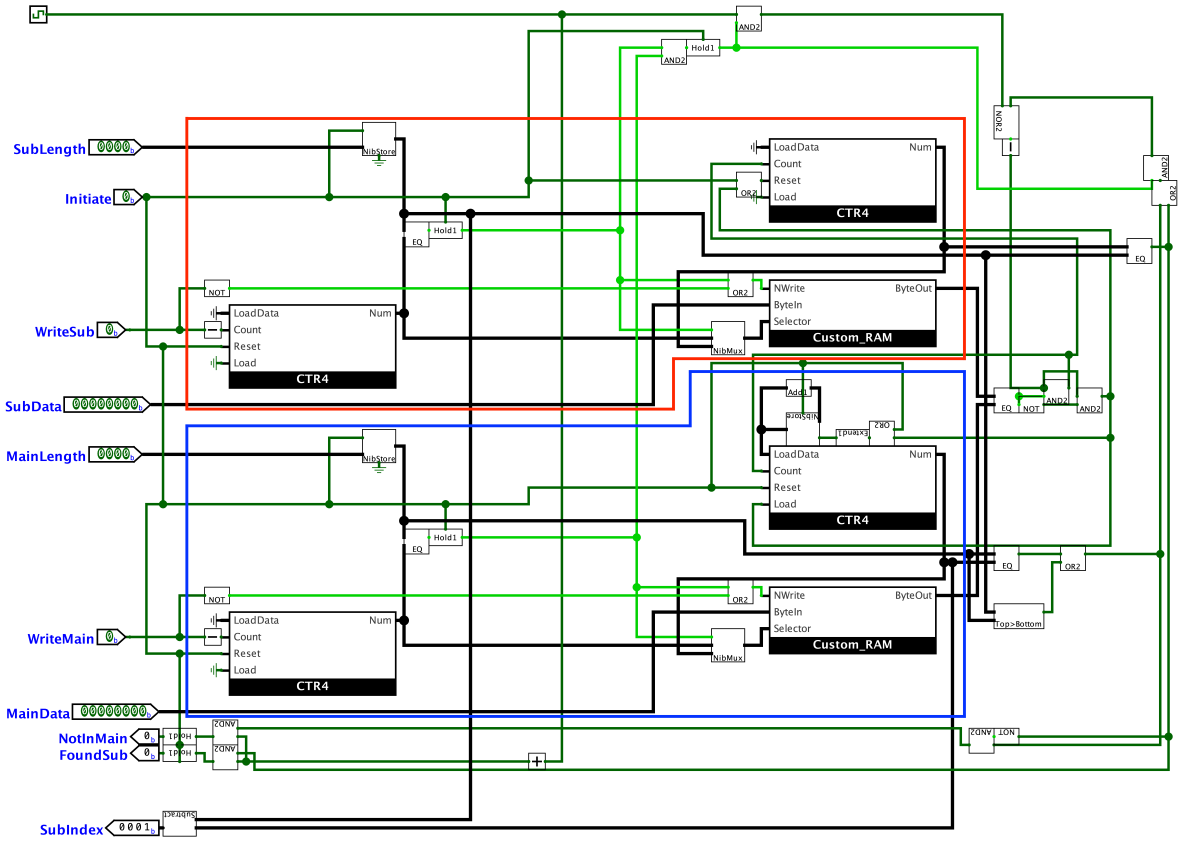

Figure 1: The completed circuit, with the regions labeled.

3.1 Intended Operation

The procedure for using the subsequence identifier is as follows: Begin by holding the subsequence length and main sequence length over the SubLength and MainLength input ports, respectively. Then, set the Initiate port to 1 for any duration and drop it back to 0. The circuit reads MainLength and SubLength until Initiate drops, at which point the circuit stores them in two NibStore blocks. Then, hold the first byte over SubData, set WriteSub to 1, and drop it to 0 to write the first byte. Then, send the second byte over SubData, set WriteSub to 1, and drop it to 0 to write the second byte. Repeat this process for every byte in the subsequence. Upon writing a number of bytes equal to the provided length, the circuit ignores all inputs to WriteSub and SubData. The process for writing the main sequence data is identical, except it uses the ports WriteMain and MainData. Writing the main sequence and subsequence are unrelated and can be done in parallel to increase efficiency. The identifier will then compare one byte per search tick until it finds or does not find the subsequence in the main sequence. At this point, the NotInMain or FoundSub output will switch to 1 and stay there until Initiate is set back to 1. When FoundSub outputs 1, the SubIndex port will hold the index in the main sequence of the subsequence's first byte. If FoundSub outputs 0, the value from SubIndex is meaningless.

3.2 Input and Output Ports

The identifier has seven input ports and three output ports. The SubLength and MainLength ports carry the length of the main sequence and subsequence, respectively, and are 4 bits wide. The Initiate port is a single bit that indicates whether to initiate a new search. The WriteSub port is a single bit and indicates whether a byte is being written from the SubData port to the built-in SRAM for the subsequence data. The SubData port is 8 bits wide and carries a single byte from the subsequence to be written to SRAM. The WriteMain and MainData ports function very similarly to WriteSub and SubData. By adding a "sending" signal for the subsequence length and main sequence length, along

with connecting all of the byte and 4-bit inputs into a single byte input and modifying the sequence length storage and counters to use bytes instead of 4-bit segments, this circuit could be made to use only a single byte bus as input. This change would then require the presence of 5 individual bit inputs, but this could also decrease to 1 by simply having a pre-specified order for the inputs.

3.3 Breakdown of the subsequence retrieval region

It consists of 4 primary blocks: a counter in the bottom left to store how many bytes have been written to its SRAM, and whose "count" input is the output of a negative edge detector attached to the WriteSub port to ensure that it only counts up once a byte has been written. In the top right is a NibStore, which stores the subsequence length and only reads when Initiate is 1. Another counter on the top left stores the subsequence byte to read from SRAM and compare it to a main sequence byte. This counter counts up whenever two bytes are equal and resets whenever they are unequal. The SRAM block in the bottom left of the region has a selector input that dictates which byte is read or written from SRAM. Due to the multiplexer that feeds the selector signal, this selector increments until every byte has been written to SRAM. After this, it transitions to a read-only state and selects from the upper-right counter until the circuit resets. The schematic detects when all the bytes have been written by running an equality check on the counter in the bottom left for bytes written and the expected sequence length held by NibStore. The equality block's output leads into a Hold1 block. Hold1 blocks remain at 0 until they receive an input of 1, at which point they remain at 1 until a reset signal rises to 1. For every Hold1 in this schematic, the reset signal is the Initiate signal.

3.4 Breakdown of the main sequence retrieval region

The main sequence retrieval block functions identically to the subsequence retrieval block, except for the counter in the upper right. Whereas the position in the subsequence should reset upon failing to match a byte, the position in the main sequence should increase by 1 for every match and add 1 to the current baseline value before reverting to it for every mismatch. The set of blocks above the counter accomplishes this. The NibStore block resets to 0 upon receiving the Initiate signal. However, since the load input also connects to the reset through an OR gate, propagation delay ensures that the reset input will drop before the load input does, thus making the NibStore load data. This data comes from Add1, a specialized block which adds 1 to a 4-bit number. Assuming the Initiate signal is held long enough, the NibStore output will be 0000; thus, the output of Add1 will be 0001. When the reset signal drops, NibStore will have enough time to store 0001 before load drops. The ExtendLong extends pulses of value 1 for eight of Logisim's iterations. If a pulse is given to some blocks containing cycles, such as the D-latch, for too little time, they will remain in a cycle and cause the simulation to stop. Since NibStore contains D-latches, a pulse extender is added to ensure it functions properly.

3.5 Breakdown of the comparison and output regions

The comparison/output region is the only clocked section of the circuit, as the circuit has not been designed with a way to know when a comparison has finished executing, and clocks only tick once every. The portion of the circuit in the top center ensures that all bytes in the main sequence and subsequence have been written to SRAM before beginning to search for subsequences. The output of the Hold1 at the top of the circuit stores whether the system has transitioned from writing to searching. Hold1's output is passed through an AND gate with the clock in the top-left to ignore all clock ticks before searching begins. This gate's output is one input of a NOR gate, whose other input exists to disable search ticks under certain conditions. The output of this NOR gate leads into a negative edge detector, which, given the preceding logic, will detect the positive edge of the circuit's clock. An EQ block is in the center of this region. The output of EQ and its complement are sent into two AND gates with the output of this negative edge detector to gate them behind a clock tick. If the two bytes are equal, the counters for both sets of bytes count up. If not, each one is set to its proper value. In the top right of this block are three gates. The EQ block's output is the "success" value. When the value of the upper right counter in the subsequence block matches the subsequence length, the circuit has located the subsequence. The success wire is sent into an OR gate with a wire representing failure to locate the subsequence, thus making the output of this gate a "finished" signal. The "finished" wire leads into an AND gate with the wire that reflects whether searching has begun

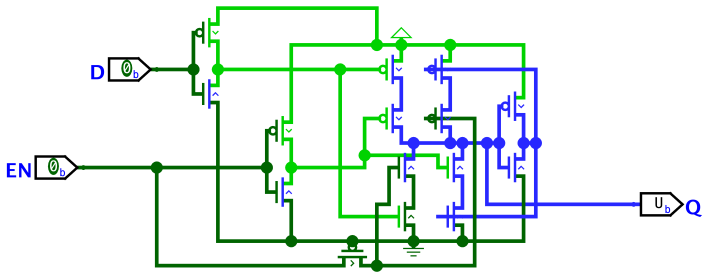

Figure 2: Circuit of the D-latch.

as a precaution against fluctuating values. The output of this AND gate is then sent into the NOR gate from earlier to force it to 0 and prevent any more search ticks, thus freezing the values in all the counters once searching is complete. The value of the failure wire is dictated by the output of an OR gate with two inputs. The first is an EQ block, which checks to see if the number of bytes read from the main sequence is equal to the length of the main sequence. This EQ block operates similarly to the EQ block that checks for success. However, all available bytes in the main sequence have been exhausted if this signal is active, so either the subsequence is not present or the circuit has found it on this search tick. The AND and NOT gates in the bottom left ensure that successful identification takes priority over failure. The Top¿Bottom comparator next to the lowest EQ block is an optimization for cases where the subsequence is longer than the main sequence. This case will result in the NotInMain signal immediately rising to 1 and preventing search ticks. In the bottom left of the schematic are some AND gates behind a positive edge detector on the clock. These are another precaution against any temporary fluctuations in the values of the success and failure wires before a final result goes to the output ports. The Hold1 blocks ensure that the value associated with the AND gates is no longer dependent on the clock once determined. The SubIndex output comes from a Subtract block and is the difference between the length of the subsequence and the position in the main sequence. This leaves the difference as the index in the main sequence of the first byte of the subsequence.

4 Breakdown of data storage mechanisms

4.1 Breakdown of the D-latch

: The custom D-latch has no logical difference from other D-latches but is more complicated due to Logisim's propagation delay. If the enable signal drops before the value of D has had at least three iterations of the simulator to update, it will enter an infinite loop until the enable signal rises again. This behavior is due to the Qbar input of the logic expression for the D-latch, which takes three iterations to propagate back to itself. If the value of Q changes within these three iterations, an unintended value will be "trapped" within the loop and oscillate endlessly. The ideal solution to this would be to include an ignore circuit, which responds to its input value changing by freezing its output for as many iterations as it takes for the loop to have the correct value. The delay transistor at the bottom of the circuit ensures that the built-in inverter for EN affects the transistors on the same iteration as EN. This design has limitations that must be accounted for by the circuits that use it. Of particular consequence is the scenario where D does not match Q, and the Enable signal is pulsed for less than three iterations, which introduces an oscillation. Storage blocks with longer oscillation periods, like Hold1, exacerbate this problem.

4.2 Breakdown of the CTR4 block

The count signal passes through a positive edge detector, which does not have an extender attached to it like many other edge detectors. This ensures that the count signal drops before the input is updated, preventing an oscillation or a double count. The reset and load signals are sent through an OR gate to create a wire reflecting whether a value should be forced into the D-latches. The NOR gates create the value to be forced in, the multiplexers force D to that value, and the output is also sent into an OR gate leading to the Enable input, forcing the D-latches to have the proper value. The XOR and AND gates

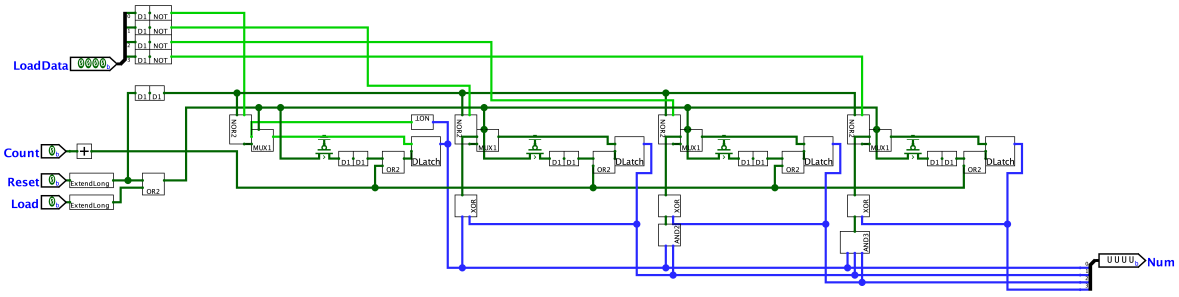

Figure 3: Circuit of the 4-bit counter.

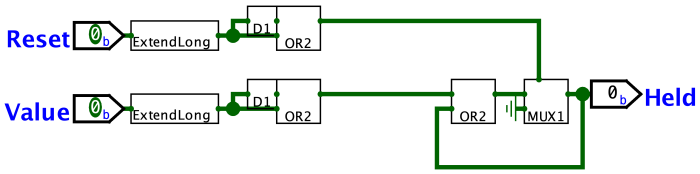

Figure 4: Circuit of the Hold1 block.

on the bottom layer of the counter and the NOT gate above the first D-latch are present to calculate the next number to load into the D-latches. The two delays on the reset wire are to ensure that the correct value of the NOR2 gate is held for long enough that the multiplexer's output will not switch while the enable signal for the D-latch is active. The two delays and the transistor connected to the OR gate feeding the Enable input of the D-latch are present to ensure Enable drops just as the multiplexer's selector signal change takes effect and possibly modifies its value. Any less delay on the Enable signal causes the multiplexer to change its value before the enable signal drops, introducing an oscillation. Any less delay on the reset wire causes the NOR2 gate's output to drop too early, eventually resulting in the multiplexer's output changing before the enable signal drops, also introducing an oscillation. The delays on the NOT inputs are not necessary for the proper functioning of the counter. However, they are present to account for the usage of the counter in the main sequence retrieval segment, where unwanted values from LoadData propagate in after the Load signal has dropped, but just before the counter has begun ignoring input from LoadData, which would result in an oscillation.

4.3 Breakdown of Hold1

The main segment of Hold1 is near the output and is a simple multiplexer connected to the output of an OR gate. The OR gate is present to implement the logic expression of the Hold1 block by looping its output into one of its inputs while still allowing for a value of 1 from the Value input to overwrite the previous value. The multiplexer is present to allow the reset signal to take effect, breaking the loop and forcing a value of 0 onto the wire when the Reset signal is 1. The number of iterations a value of 1 must be held on the OR gate or multiplexer selector without introducing an oscillation is 12. However, due to the situational uses of the Hold1 block, the extenders for the Reset and Value inputs are incorporated directly into Hold1.

4.4 Breakdown of ByteStore

The ByteStore is a simple block that takes a byte as input and stores each bit in a separate D-latch, which connects to the byte output. Each D-latch has its enable signal connected to a single enable input, making the ByteStore a direct extension of the D-latch to 8 bits.

4.5 Breakdown of the SRAM

The length of the sequences was mainly limited to 4 bits instead of 8 bits due to the tedium of scaling up the number of storage blocks, scaling up the logic expression in the Decoder, and the speed limitations of Logisim. At only 16 items, Logisim takes approximately 10 seconds to relocate an SRAM block. To

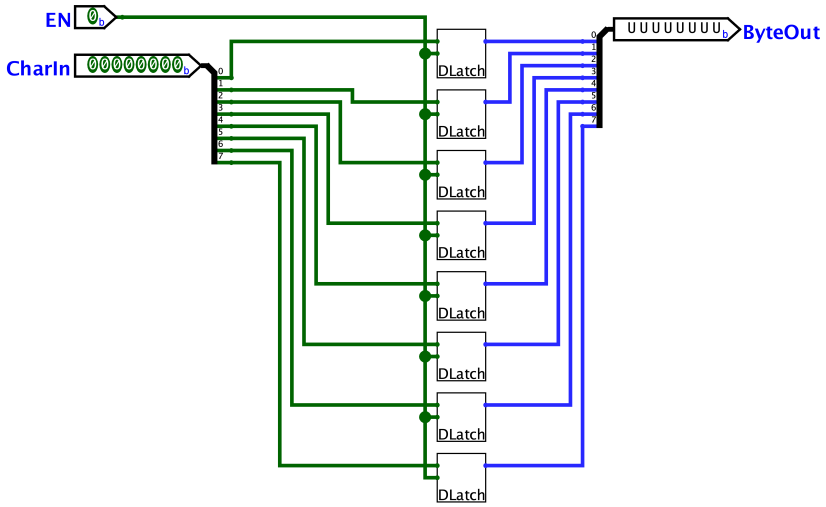

Figure 5: Circuit of the ByteStore block.

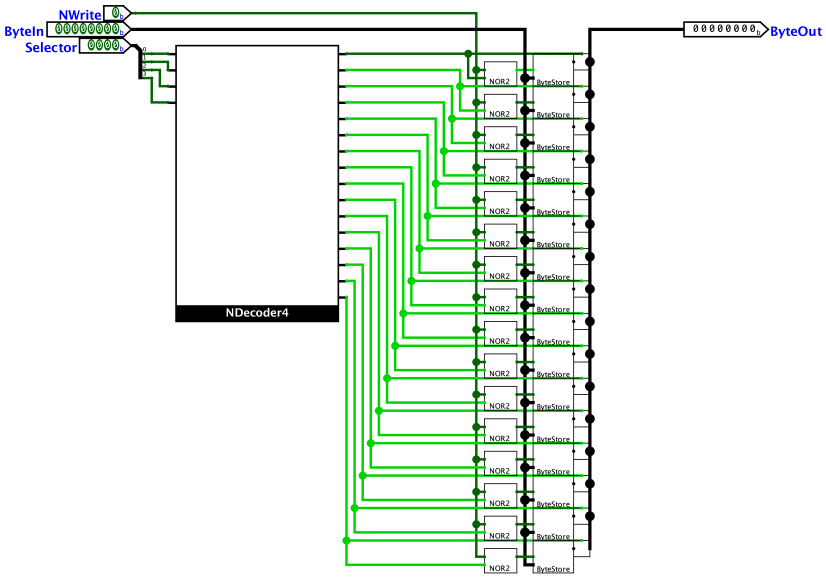

Figure 6: Circuit of the 16-byte SRAM.

expand it to have 256 cells would greatly slow down the speed of Logisim and the speed of design. The RAM block consists of 2 segments: the Decoder and the repeated storage blocks. The Decoder's output is the inverse of the value encoded by the input. This inversion allows inverted gates in the repeated storage blocks, which speeds them up. The repeated storage blocks consist of a NOR gate with the NWrite input and Decoder output as its inputs, which makes the output of the NOR gate indicate whether a cell should accept data. The cell's output is sent into the enable inputs of the ByteStore blocks, ensuring they only load in the data sent to them when necessary. This allows one byte to connect to the data inputs without an individual input or a byte transmission gate for each. There is no output enable/disable for the ByteStore blocks, so unlabeled byte transmission gates are placed on the outputs of the ByteStore. The "TransmitBar" inputs of the transmission gates are connected to the Decoder's output, thus making only the ByteStore selected by the selector input send its data to the output. The ByteStore outputs do not have the value 0 when not transmitting, but rather an undefined value which any well-defined value will override. Exactly one byte transmission gate will output at any given time, so the wire will always have exactly one value. Note that the NWrite input is not a read/NWrite input; it is only an NWrite input. SRAM continuously outputs data, even as new data loads.

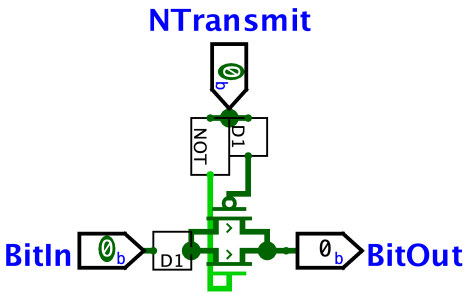

Figure 7: Circuit of the single-bit transmission gate.

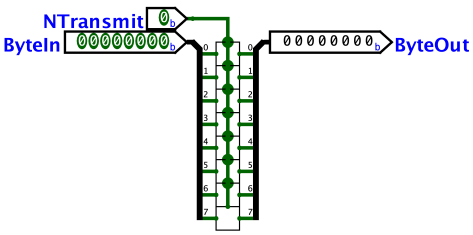

Figure 8: Circuit of the byte transmission gate.

5 Breakdown of the other miscellaneous blocks

5.1 Breakdown of the Transmission Gates

The most basic transmission gate is the Bit transmission gate. Its output is undefined when the NTransmit input is 1 and reflects the input from BitIn when NTransmit is false. The delay on the input is present to ensure that the NTransmit and BitIn inputs propagate to the output simultaneously, ensuring that the output logic expression is precisely that of a transmission gate and not dependent on past values of NTransmit, as would be the case if the delays were not present. Only one of these transistors is required in Logisim, though both are present for consistency with traditional MOSFET-based transmission gates. The byte transmission gate is simply 8 bit transmission gates in parallel, with a shared NTransmit signal. A nearly identical 4-bit transmission gate exists but has a 4-bit input and output instead of 8-bit inputs and outputs.

5.2 Breakdown of the Decoder

The Decoder is relatively simple, but its outputs are all inverted for efficiency. Each of the four inputs is inverted to generate four pairs of wires. By choosing one wire from each pair and passing them through a NAND4, the logic expressions for each of the 16 decoded outputs can be generated in parallel. The delays below each inverter ensure that changes in the uninverted wires are synchronized with the inverted wires.

5.3 Breakdown of the basic logic gates

Basic logic gates, like the OR, AND, NAND, NOR, and NOT, follow standard CMOS implementations. However, they all have additional meaningless transistors that modify their logic expressions to account for propagation delay in the transistors. Sequentially linked PMOS transistors, such as those in a NOR gate, require that their accompanying NMOS have a delay equivalent to the number of PMOS transistors after them in the sequence. This prevents an erroneous value when an NMOS transistor can pull the output to 0 before the PMOS transistors have stopped pulling the output to 1, sending an error signal along the output for 1 iteration before settling on the correct value. Note that because of how Logisim simulates, sequential transistors modified in this way do not implement the correct logic expressions. Their true logic expressions are a mix of values from different iterations due to the propagation delay introduced by the sequential transistors. For example, a logical NOR gate does not output the NOR of its two inputs. It outputs the logical NOR of its current lower input value and the previous iteration's upper input value. When both inputs begin at 0, and the circuit is stable, if both inputs are pulsed with a value of 1 for 1 iteration simultaneously, the output of the NOR gate will drop to 0 for two iterations instead of 1. The presence of the meaningless transistors before the PMOS transistors resolves this issue by introducing another delay equivalent to the delay of all the preceding PMOS transistors, synchronizing all the inputs and outputs. This synchronization makes the delay of inverting gates very easy to calculate. An inverting gate with N inputs must have an internal propagation delay of $N + 1$ iterations, as there must be enough time for the inputs to travel down the longest possible sequential chain of transistors, which is N transistors long. The additional

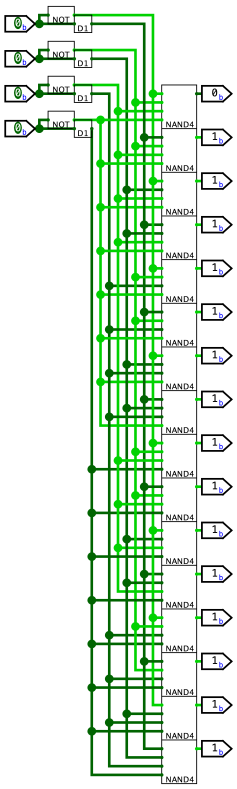

Figure 9: Circuit of the 4 to 16 decoder.

1 is due to the one iteration delay while the inputs influence the transistors' gates. When embedded in another block, since there is a 1 iteration delay associated with entering and leaving a block, the delay is $N+3$ iterations. The use of PMOS transistors rather than NMOS to delay is arbitrary, as an NMOS whose gate connects to power would function identically to a PMOS whose gate connects to ground. Only figures for the NAND4 and NOR4 gates have been provided here, as they are representative of the other logic gates.

5.4 Breakdown of the XOR gate

The XOR gate consists of two main blocks: a NOR gate and a NAND gate. Each of these blocks is individually constructed in the same way as in the other gates, notably with all the delay transistors intact. To convert this pair of gates into a single XOR gate, a PMOS transistor is placed, with its source being the NAND gate and its gate being the NOR gate. Since the logic expression for the output

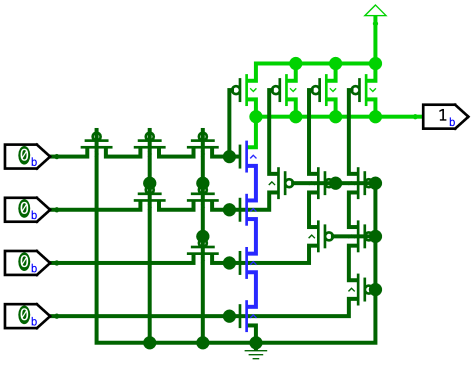

Figure 10: Circuit of a 4-input NAND gate.

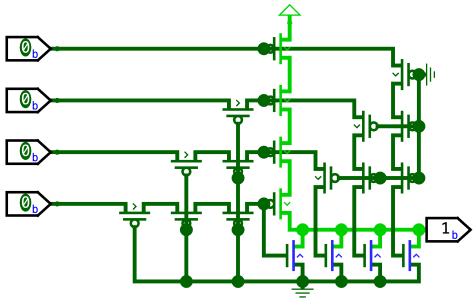

Figure 11: Circuit of a 4-input NOR gate.

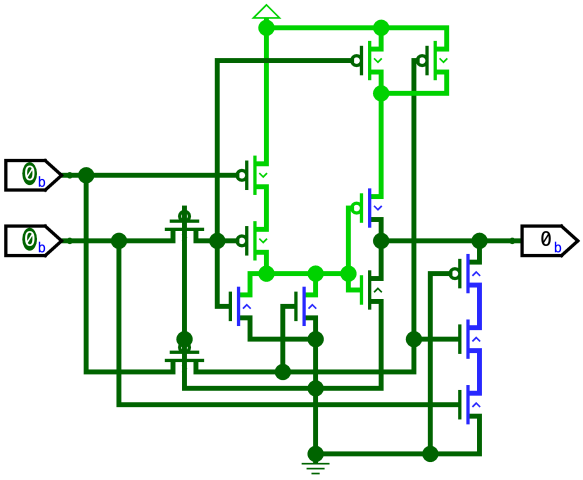

Figure 12: Circuit of the XOR gate.

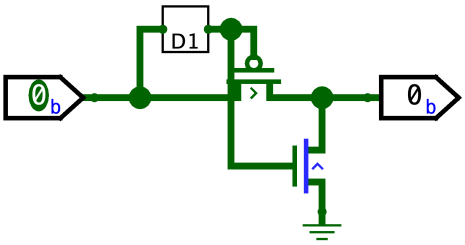

Figure 13: Circuit of the positive edge detector.

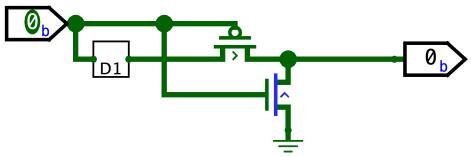

Figure 14: Circuit of the negative edge detector.

of a PMOS transistor is $Source \wedge \neg Gate$, this implements $\neg(A \wedge B) \wedge \neg\neg(A \vee B)$, which simplifies to $\neg(A \wedge B) \wedge (A \vee B)$, which is $A \oplus B$. If an NMOS transistor is added along with the PMOS transistor, with its gate being the NOR gate output, its source being ground, and its drain connecting to the drain of the PMOS, the logic expression for when the output is low is $\neg(A \vee B)$. To convert this to the desired $\neg(A \oplus B)$ logic expression, the $A \wedge B$ logic from the NAND gate's sequential NMOS transistors must connect to the output wire. The additional delay transistor after the two sequential NMOS transistors is present to account for the delay introduced by the PMOS transistor in the middle of the circuit.

5.5 Breakdown of the edge detectors

Both edge detectors are simple. For the positive edge detector, when the input is 0, the output is 0. When the input becomes 1, it leads to the source of a PMOS transistor. This transistor's gate is 0, so the output of the circuit is 1. The input also leads to a delay of three iterations, the output of which leads to the inputs of the PMOS transistor and NMOS transistor. This ensures that the connection of the input to the output will be severed after three iterations, and the NMOS to ground ensures that the output drops to 0 after this. The negative edge detector is similar, but the delay is placed on the source input of the PMOS rather than on the PMOS and NMOS gate inputs. This inverts the delays and logic expression, making it a negative edge detector rather than a positive one.

5.6 Breakdown of the multiplexer

The multiplexer is another simple block. Its behavior was already discussed in the segment on the SRAM, where a series of wires are connected such that only one of the sources has a value and the rest are left undefined, so the properly valued source overwrites all the undefined wires. The selector input is sent directly to one of the bit transmission gates to open and close it as needed, and then inverted

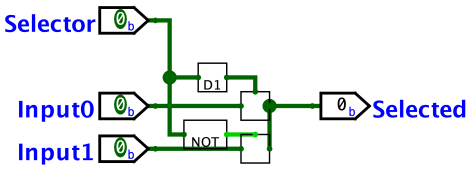

Figure 15: Circuit of the single-bit Multiplexer.

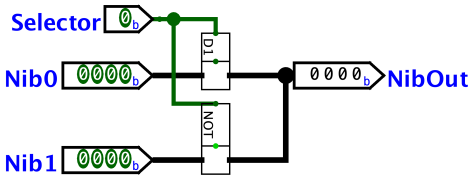

Figure 16: Circuit of the 4-bit multiplexer.

and sent as input to the other transmission gate to select if and only if the first is not selected. A delay is attached to the upper multiplexer to synchronize changes in its NTransmit signal with the NTransmit of the bit transmission gate whose input is inverted. Note that because the non-selector inputs lack any delays, this multiplexer is not selecting from the current values of the non-selector inputs but rather values of these inputs in previous Logisim iterations. This behavior is undesirable but was kept as it does not affect the identifier's function, and resolving it would require many changes to other blocks that were designed around this mismatch. Minor delays in the inputs would resolve this issue.

5.7 Breakdown of the Arithmetic blocks

The Equality blocks are also simple: Each pair of bits of the same significance enters an XOR gate, and the outputs of all the XOR gates enter a NOR gate. The 4-bit subtractor is much more complicated. The diagram does not make it clear, but this is a series of 4 full subtractors with all unnecessary elements removed. The logic expression for a full subtractor's Difference output is $(Borrow \oplus Minuend \oplus Subtrahend)$, and the logic expression for its Borrow output is $(\neg((Borrow \oplus Minuend) \vee (\neg Borrow))) \vee (\neg(Minuend \vee \neg Subtrahend))$. However, outputting $\neg Borrow$ allows for the use of a NOR gate, which is done to improve speed. The unnecessary elements are the Borrow input for the first bit and the Borrow output for the last bit. Since the borrow input for the first bit is 0, the logic expression for its Difference output can be simplified to $Borrow \oplus Minuend$, and its Borrow output can be simplified to $Subtrahend \wedge \neg Minuend$. Since the Borrow outputs are inverted, the $\neg Borrow$ output is $\neg(Subtrahend \wedge \neg Minuend)$, which once again increases the speed of the subtractor as a NAND gate. Since this element's only use in the circuit is to subtract the length of the subsequence from the position in the main sequence, the minuend must be larger than the subtrahend. This means that the Borrow output for the last bit must always be 0, so the logic expression for the Difference output can be simplified to $Minuend \wedge \neg(Subtrahend \vee Borrow)$. The NibAdd1 implements a series of 4 Full adders when one of the addends is 1, again simplifying unnecessary carry-in and carry-out bits. The NibGreaterThan block only needs to check if the most significant bit of the first number is greater or less than the most significant bit of the second number. If the first number's most significant bit is greater than the second number's most significant bit, the first number is larger than the second. If the second number's most significant bit is greater than the first number's most significant bit, then the first number is smaller than the second number. If the bits are equal, the process will be repeated with the next most significant bit. Since differences in more significant bits take priority over differences in less significant bits, calculating whether there is a difference in less significant bits requires the absence of a difference in more significant bits, which is accomplished with XOR gates and the increasingly large NAND gates that pull from every more significant XOR gate before it can reflect whether one bit is larger or smaller than another.

6 Applicability

Though applicability was not a concern for this project, there are some potential niche applications for this technology if it is further refined. The human genome is exceptionally long, which makes searching the entire genome for a subsequence without knowing its general location ahead of time extremely time-consuming. This circuit seems particularly well-suited to the task.

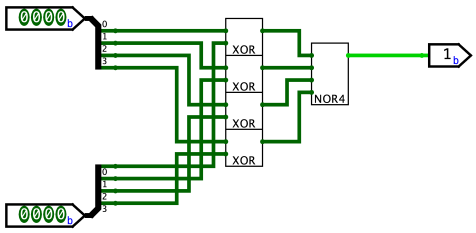

Figure 17: Circuit of the 4-bit equality checker.

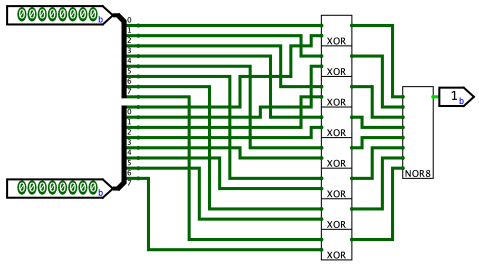

Figure 18: Circuit of the byte equality checker.

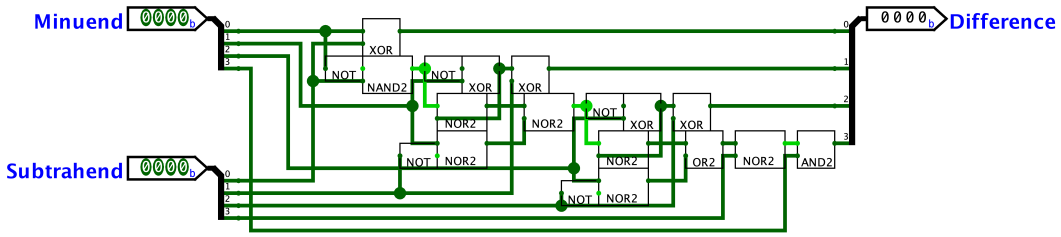

Figure 19: Circuit of the 4-bit subtractor.

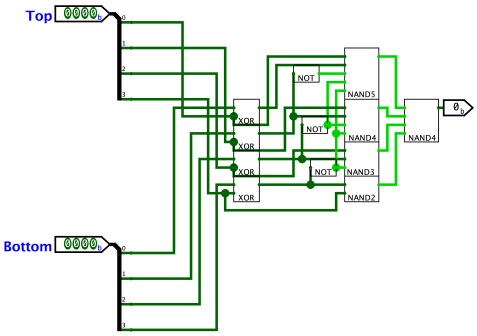

Figure 20: The 4-bit comparator, called NibGreaterThan.

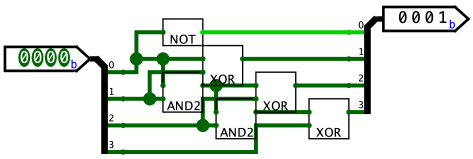

Figure 21: The 4-bit incrementer, called NibAdd1.