Project Manual: Analog Dino Game

Jay Tang, Pritom Paul, Adam Todd

LTSPICE FILES

Description

This project was inspired from the dinosaur game that appears whenever you try to access a webpage on Google Chrome without an internet connection. In this game, the player is a T-rex that continuously runs across a black-and-white landscape full of obstacles by jumping. As the game progresses, the speed gradually increases until the user hits an obstacle, resulting in a game over. Our group decided to make a computer that runs an analog version of this game. Our version of the game plays on a small array of 4 LEDs. The leftmost LED indicates the player, while the right 3 LEDs represent possible positions of obstacles. All obstacles will start at the right most LED and move towards the left LED by LED. To avoid an obstacle, push a button when the obstacle is at the 2nd left most LED. If you fail to, your player LED will turn off and the game ends. Obstacles will come at set intervals of time. Milestone 1 focuses on implementing the jumping and timing aspect of the game, which is done via an astable multivibrator and AND gate. This ensures that players can only jump at specific times and will stay in the air for exactly one cycle. Milestone 2 focuses on creating memory for the game using CMOS logic and electronic switches. This creates the gameplay loop, where obstacles move across the screen without player inputs. Milestone 3 focuses on taking the previous parts of the game and integrating them to create a seamless gameplay experience. A block diagram containing the elements used in the milestones can be found in Figure 1, Figure 2, and Figure 3 on the next page.

Block Diagram:

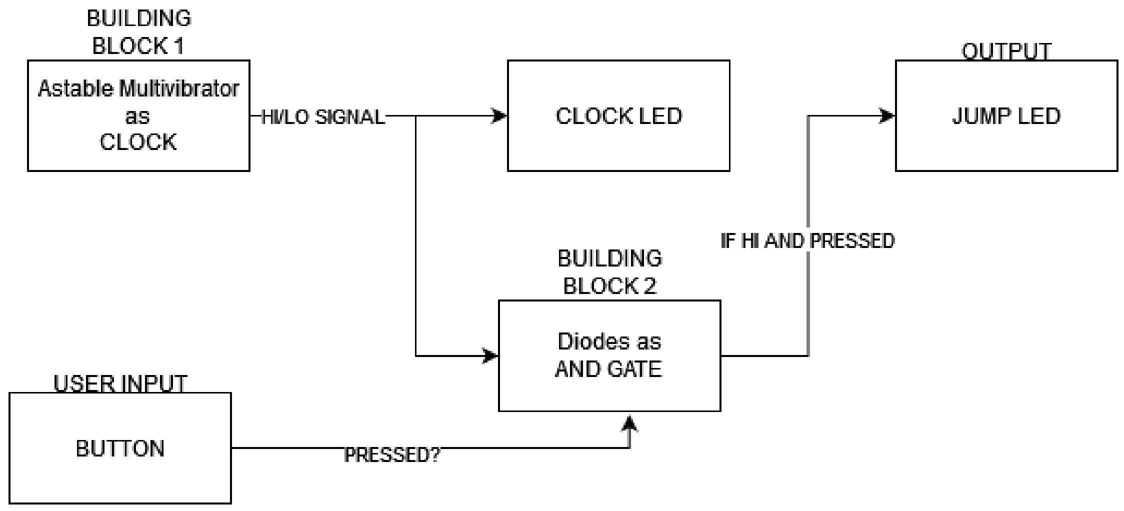


Figure 1: MS1 Block Diagram

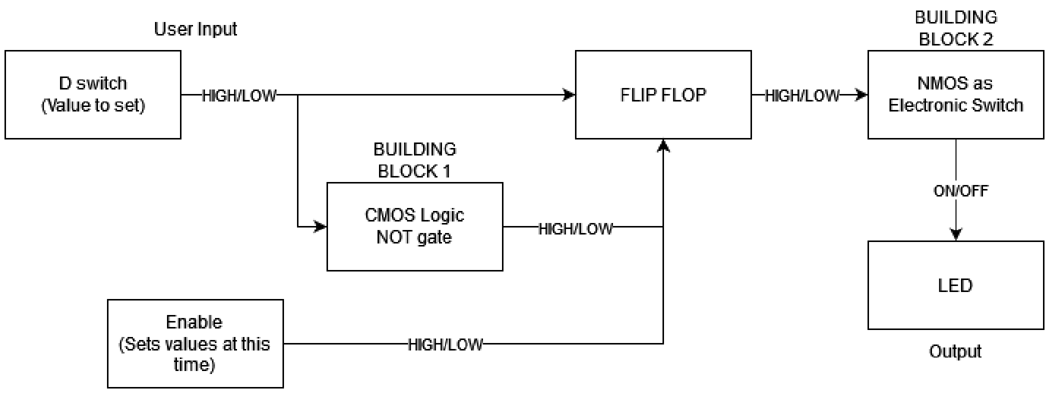


Figure 2: MS2 Block Diagram

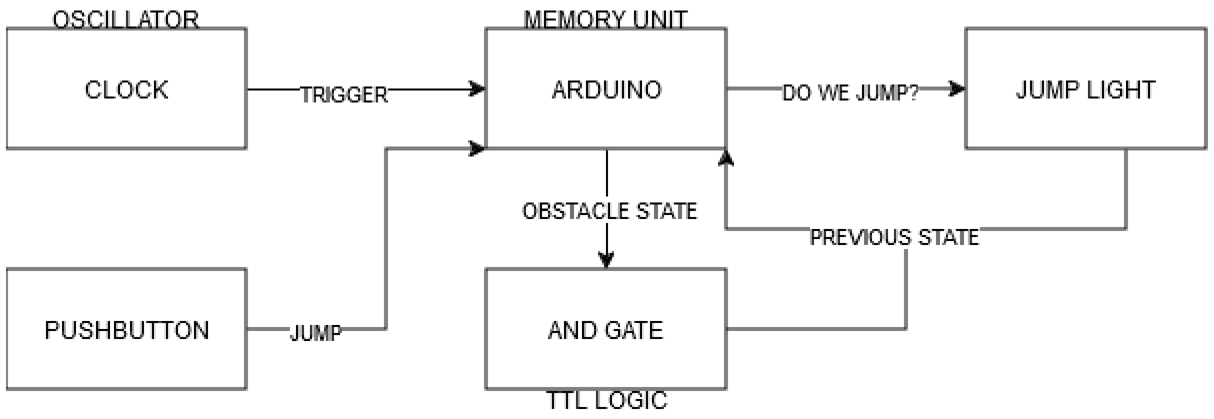


Figure 3: MS3 Block Diagram

Schematic:

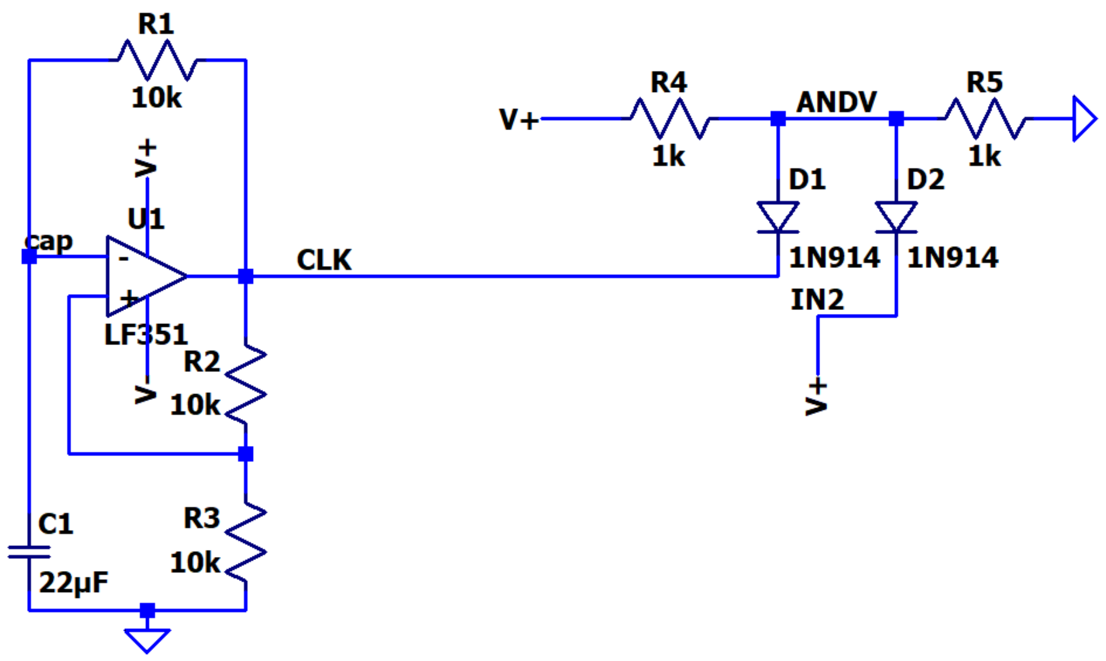


Figure 4: LTspice schematic of the entire Milestone 1

Figure 4 starts with an astable multivibrator on the left that has an output labeled CLK. This feeds to our diode AND gate on the right, composed of D1 and D2. V+, a 5V supply, is connected through R5, our load in this case. V+ The schematic does not contain LEDs, which aren't accurately represented in LTSpice. The CLOCK LED has no replacement, while JUMP LED is replaced by R5.

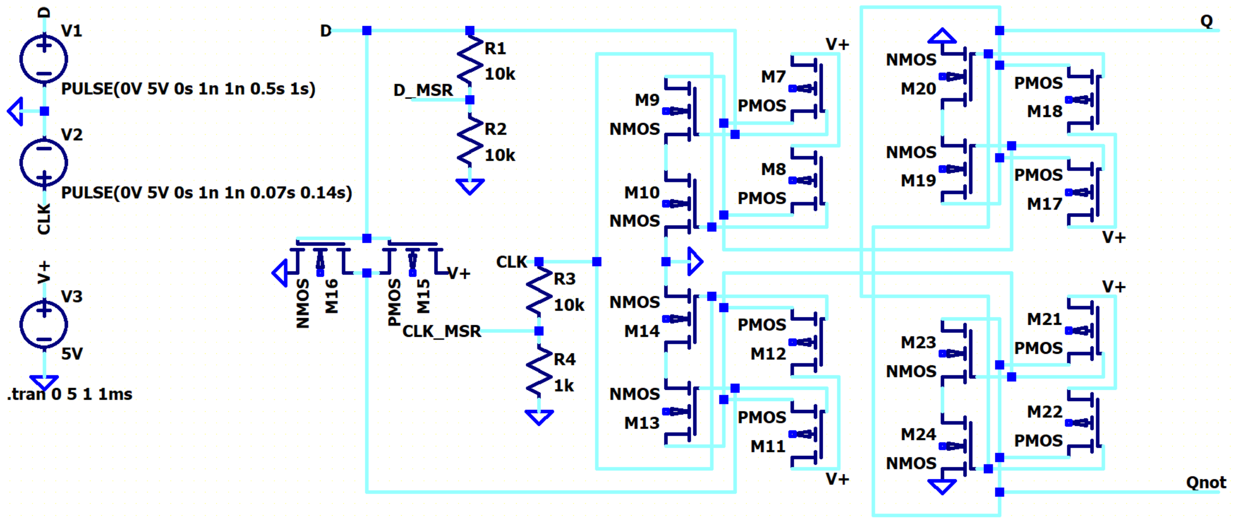


Figure 5: LTSpice schematic of entire Milestone 2

Figure 5 shows multiple CMOS logic gates, which are fed with an input D and an enabled clock CLK. They output Q and Qnot. The electric switch and LED are not represented on the LTSpice schematic, as simply probing Q will get the necessary data points.

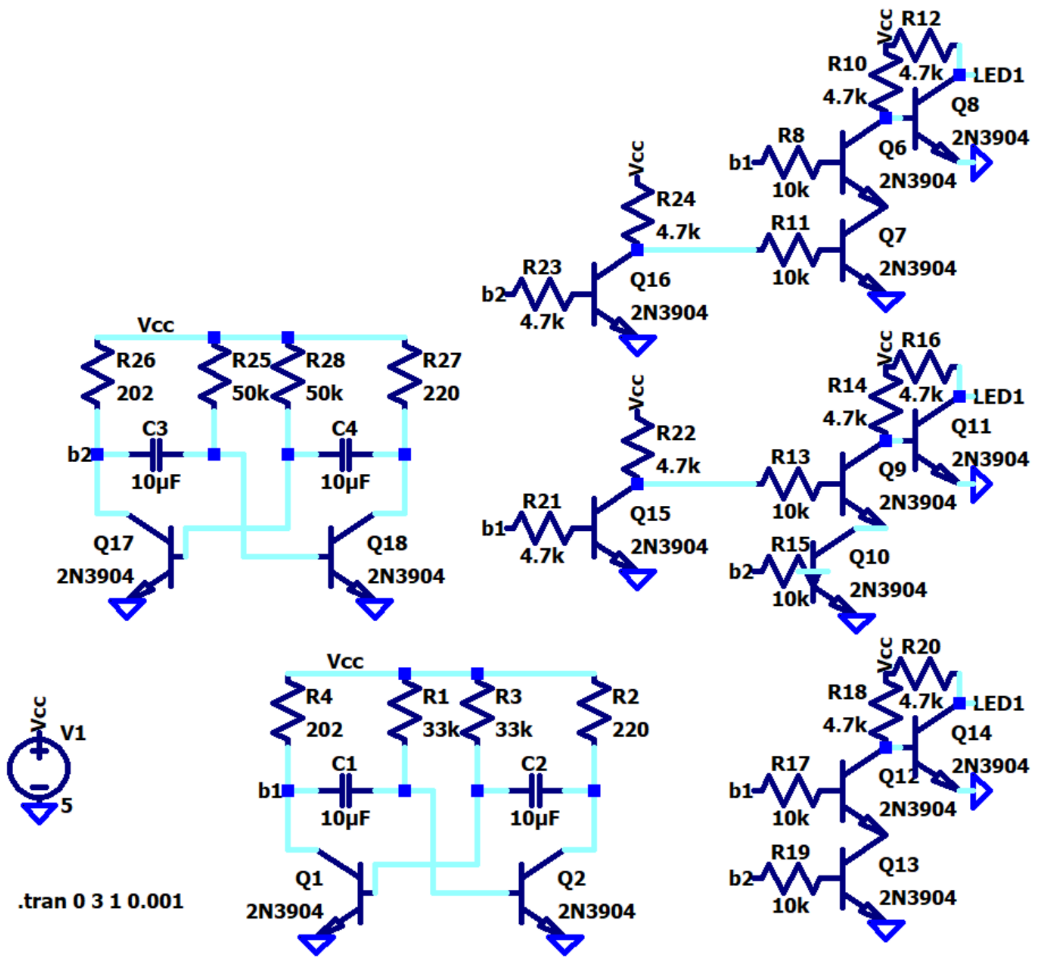


Figure 6: LTSpice schematic of entire Milestone 3

Figure 6 shows multiple AND gates constructed from a NAND and a NOT made using RTL logic with BJT transistors and resistors. Vcc indicates the clock signal, and LED the LEDs that will be lit up by the signal

High Level Description of Application

Figure 4 implements a jump switch to be used for a side scrolling platformer game. The command to jump is sent by the flick of a switch. To ensure that the game has time to process the jump, the jump command can only be sent when the clock signal is HIGH. This is done using an AND gate. The AND gate only sends the jump signal through when both the clock signal is

HIGH and the switch is in the ON position. A CLOCK LED exists to show that the clock is in its HIGH state and that flicking the switch ON will send the jump command. A JUMP LED exists to show that the jump command is received.

Figure 5 implements memory. During the high clock period, Q follows D. When the clock cycles low, whatever state Q was in remains locked until the clock cycles high again. This ensures things are always the same while our circuit is calculating the next state based on the current state. Input D is currently controlled by a switch, but will later be voltage from the previous clock cycle. An LED is attached to output Q to show when the Q state is high.

Figure 6 shows the entire game, a series of AND gates and a clock. It is simpler than the previous milestone as most of the complicated circuitry was moved to Arduino code. Each and-gate receives input from which light was on previously as inputs, and from there outputs the next light that should be on. These lights represent obstacles. If light 1(only the first and gate) was on previously, then light 2(only the second and gate) will be on next. if light 2, then light 3, and so on and so forth in a cycle. Each state switch happens on a rising clock edge where all of the states held during the falling clock get checked and processed to the next state. While this is happening, a button where you can light up a jump LED lays on the side, ready to make you jump over an oncoming obstacle. The game checks repeatedly if LED3 is on, and if you jumped while LED3 was on. If you didn’t, you would lose and the clock signal would stop being read.

Description of Related Pre-existing Applications

In order to synchronize things, computers take inputs at specific intervals. This input is then put into a buffer, where the inputs will be processed and their corresponding outputs given. To perform the actual logic that maps inputs to outputs, most home brew computers utilize transistor to transistor logic. Transistors are able to switch on and off at very high speeds, allowing for many operations a second. This is important for complex logic that may require multiple operations. Afterwards, the on and off states of different combinations of transistors are used to indicate states. These states are mapped to outputs by enabling or disabling specific wires in an output bus that usually connects to a screen that displays information based on the transistor states. The transistors are usually configured as edge-triggered bistables, otherwise known as flip-flops/latches. These things contain memory, meaning they will store the same state unless an input is applied. A clock, usually in the form of a dedicated clock IC is used to provide a consistent interval for the transistors to check if they need to change states. In our case, our “screen” is our 4 LED array that represents the positions of obstacles and our player.

Astable Multivibrator

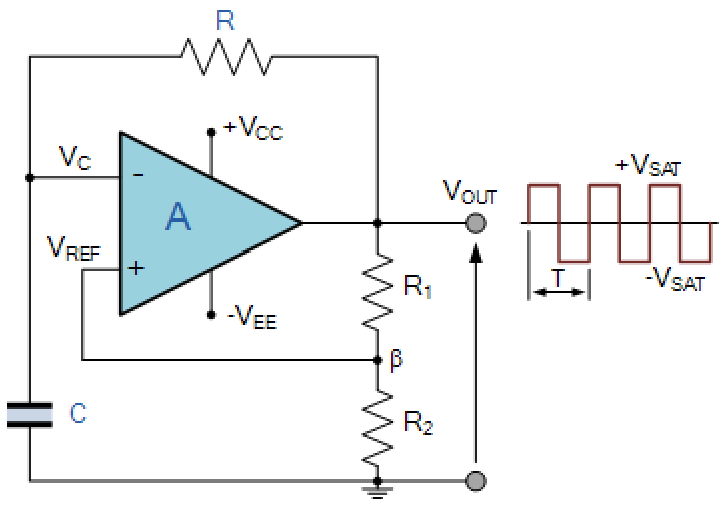


Figure 7: Example Astable Multivibrator

A multivibrator generates a square wave that can be used to provide periodic input. For the purpose of this project, the astable multivibrator (MV) is utilized to make a clock. The MV alternates between high voltage and low voltage, with each rise into high voltage being considered 1 tick of our clock. This helps sync our game, ensuring any and all events happen consistently.

LF351 Op-Amp

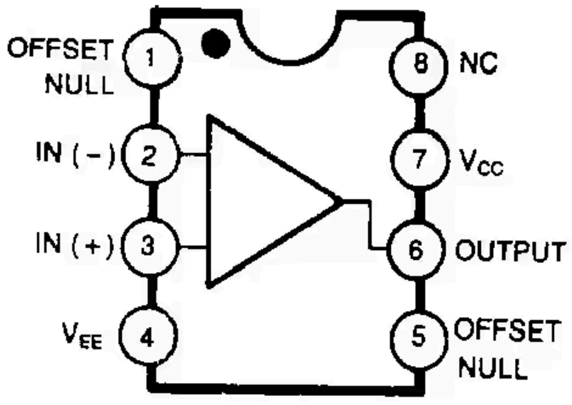


Figure 8: LF351 Datasheet Schematic

The op-amp of choice for this lab is the LF351. It is configured as an astable multivibrator, meaning it transitions between 2 states at constant time intervals. The LF351 was chosen because of its high slew rate of 13V/µS, enabling us to have more responsive changes.

Diode-based Logic AND-gate

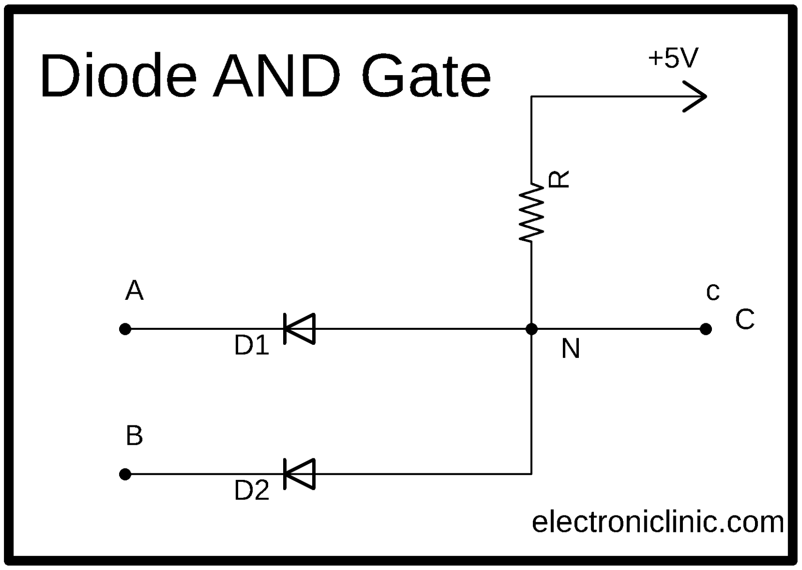


Figure97: Diode AND-Gate schematics

Figure 9 shows an example of what a Diode AND-gate would look like isolated from our main circuit. This circuit would contain a power supply and 2 diodes that are connected in reverse biased configuration would allow current to flow forward. We chose to do a logic AND because it has to be configured with our input stage where when pressed, it can only send an output when both clocks are high and are pressed. Diode AND-gate works as any and-gate logic with its truth table, which this design choice will enable our MS1 game logic.

1N914 Diode

Figure 10: Small Signal Fast Switching Diodes

Figure 10 shows a type of diode that we will be using which is the 1N914 diode. It was chosen because of its availability in our kit and the fact that it met all our basic requirements of acting as a one way switch.

2N7000 Transistor

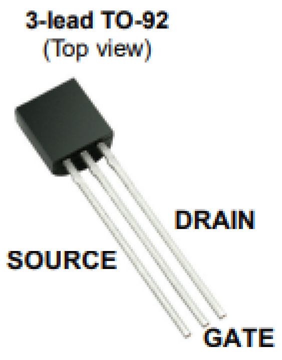


Figure 11: 2N7000 N-channel MOSFET

Figure 11 shows the type of transistor that we will be using which is the 2N7000 transistor. It was chosen because of its availability in our kit and meets the requirements that are needed for parts of our logic of the overall circuit.

ALD1105 Transistor

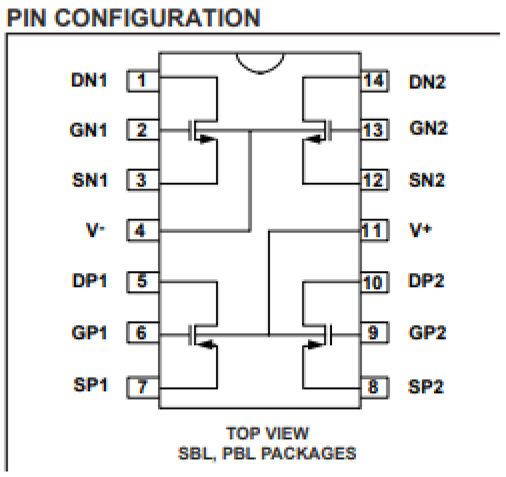


Figure 12: ALD1105 Dual CMOS chip

Figure 12 is the internal schematic for the ALD1105 dual cmos chip that we used extensively throughout this milestone. Its 2 NMOS and 2 PMOS configuration is perfect to fit one NAND gate per chip.

CMOS-based Logic NAND-gate

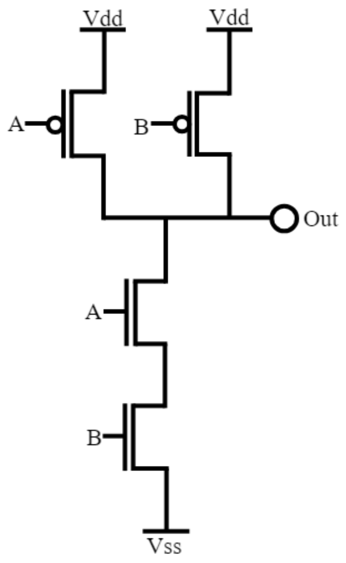


Figure 13: CMOS NAND Gate

Figure 13 shows an example of what a CMOS NAND-gate would look like isolated from our main circuit. This circuit would contain 2 power supply (Vdd) and a couple of mosfets. We chose to do a logic NAND because it has to be configured with our stages to handle the game logic as well as the memey stuff from the flip flop. The NAND gate works with our design due to its truth table for MS2.

CMOS-based Logic NOT-gate

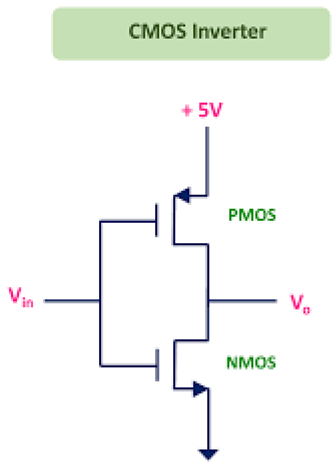


Figure 14: CMOS Inverter

Figure 14 shows an example of what a CMOS Inventor (basically NOT-gate) would look like isolated from our main circuit. As shown from the figure, the input is applied to the gate terminal of the two CMOS transistors, and the output is connected to their drain terminals. We chose to do this because as a way to communicate between the user inputs stage and the memory.

2N3904 NPN Transistor

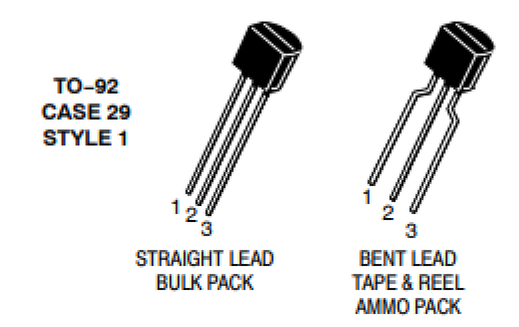


Figure 15: BJT transistor

Figure 15 is the only BJT transistor that is needed for the circuit as it is a semiconductor device that operates with charge carriers, holes and electrons to amplify our circuit. We are mainly operating in NPN junctions so we can work with positive voltages from the collector to the Emitter.

RTL Logic Gate

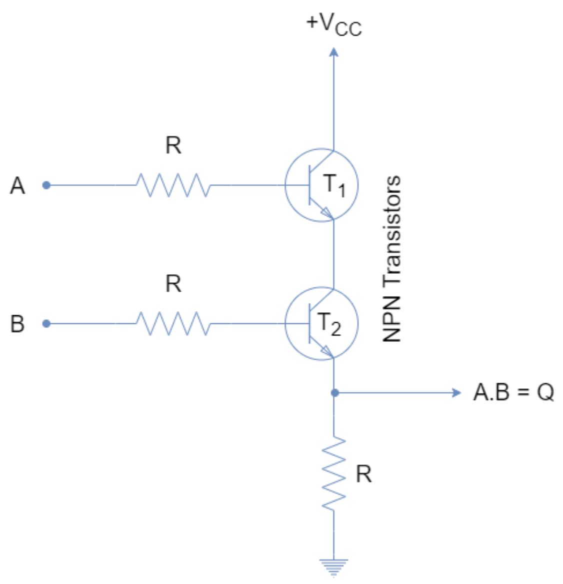


Figure 16: RTL AND Gate

Figure 16 is our RTL logic gate that we are using for the MS3 building block. We continue to work on the AND logic gate verison of the RTL logic because it can be used to control our game logic in our game plan as well as states. The advantage of the RTL logic gate compared to our options is that it uses less transistors and it's simple to use. The RTL AND gate would be the same truth table of a regular and-gate where both inputs are HIGH in order for the output to be ON.

Arduino Uno

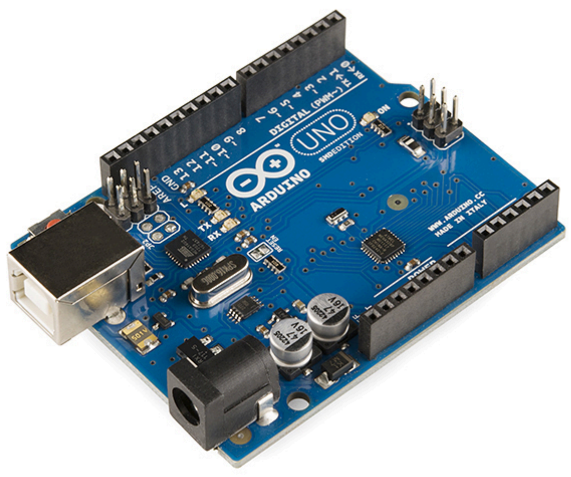


Figure 17: Arduino Uno

Figure 17 is the arduino uno that is being used for the circuit as a way to store memory. Due to time constraints, we are unable to implement the necessary components like adding a bunch of flip flops. Arduino coding is programmed in c code (refer to the Appendix) and the documentation for how to use the arduino is in the references.

Operation and Design

Input Block

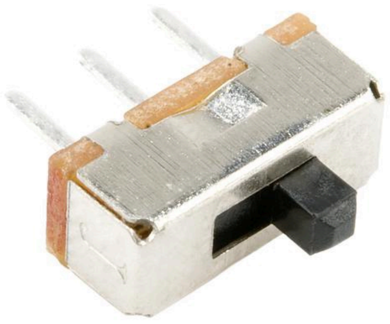


Figure 18: Side Switch image (not the same switch but similar concept)

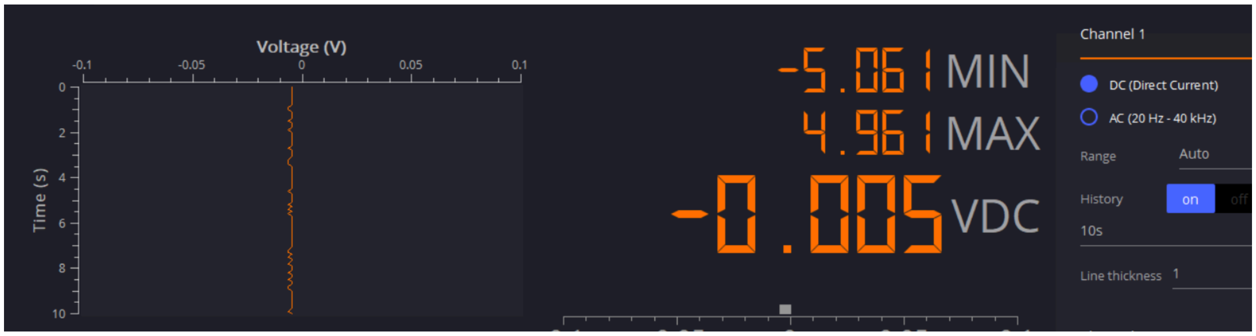
Description

The input for the system is a switch that the user will flip. When the switch is in the ON state, an output (voltage) will be at approximately 5V. When the switch is in the OFF state, the output will be at approximately 0V.

Input/Output Plot

Figure 19 and *Figure 20* shows the voltage output from the switch within the entire circuit connected.

When the switch is in the OFF state, we measure:

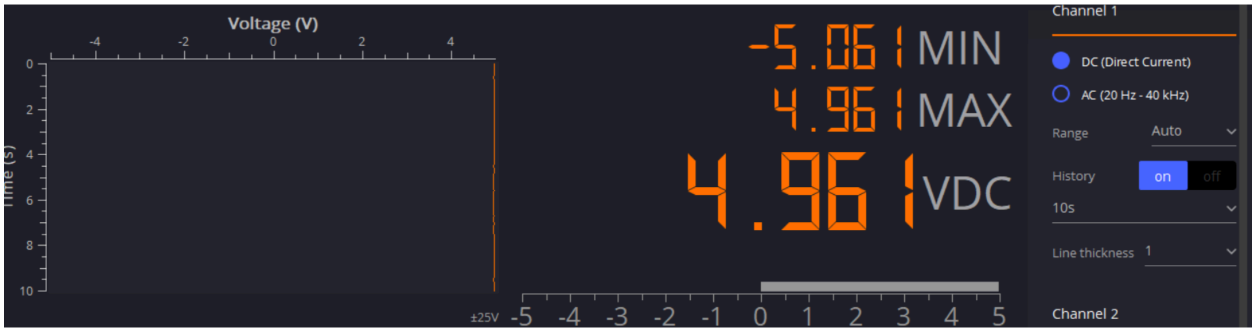


-5.061 MIN
4.961 MAX
-0.005 VDC

DC (Direct Current)
AC (20 Hz - 40 kHz)
Range Auto
History on off
10s
Line thickness 1

Figure 19: Switch state: switch off

When the switch is in the ON state, we measure:



DC (Direct Current) ●
AC (20 Hz - 40 kHz)
Range: Auto
History: on
10s
Line thickness: 1

Figure 20: Switch State: switch on

MS1 Building Block 1: Astable Multivibrator (MV)

Schematic

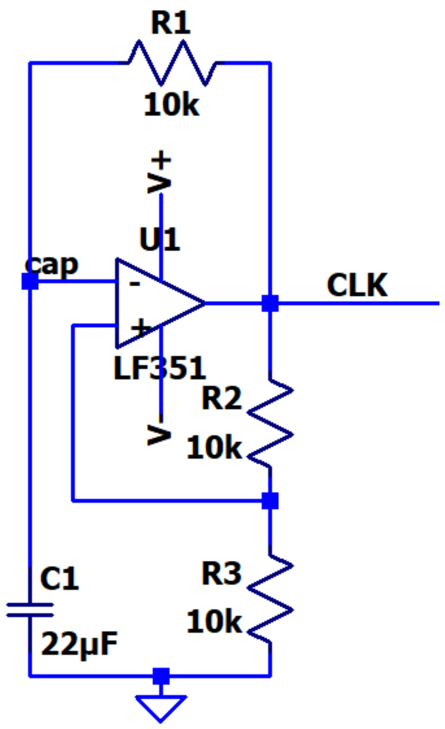


Figure 21: Isolated Astable Multivibrator

Design Equation

In a MV circuit, the feedback fraction β which is the voltage divider in the feedback loop is given:

$$\beta = \frac{R_2}{R_1+R_2} \#(1)$$

To elaborate, the output waveform's period is determined by the RC time constants of the two timing components, as well as the feedback ratio established by the R1, R2 voltage divider network, which sets the reference voltage $(V_{sat+} \& V_{sat-})$.

$$V_{sat+} = \beta * V_{cc}$$

$$V_{sat-} = \beta * V_{ee} \quad \#(2)$$

If the positive and negative values of the amplifier's saturation voltage are the same magnitude, then V_sat+ = V_sat-, then expression to calculate the period of oscillation becomes:

$$T = 2RC * ln(\frac{1+\beta}{1-\beta}) \#(3)$$

Using equation #1, #2, and #3 to solve for the frequency of the MV results us to have our ideal value of:

$\beta = \frac{10k}{10k+10k} = 0.5$

$V_{sat+} = 0.5(5V) = 2.5V$ & $V_{sat-} = 0.5(-5V) =- 2.5V$

$T = 2(10k)(22\mu F) * ln(\frac{1+0.5}{1-0.5}) = 0.48s$

$f = \frac{1}{T} = \frac{1}{0.48s} = 2.08Hz$

Discussion of Component Choices

We first started out by choosing a β value. For the sake of convenience, we used two $10k\Omega$ resistors to get a β of 0.5. This gives us approximately 1.09, which simplifies the equation to $\frac{1}{2.18RC}$ From here, we chose our R and C values to reach our target value of 2 Hz. We once again choose 10k to be our resistor value, and that leaves us with 22µF as our only available capacitor choice.

Input/Output Plot

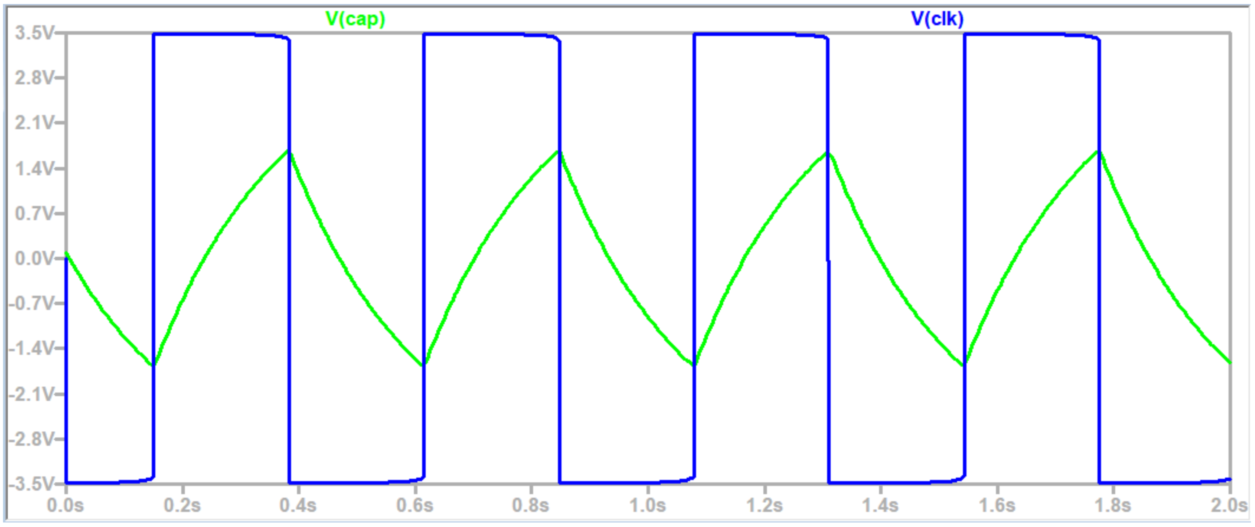


Figure 22: Astable Multivibrator Output

In *Figure 22,* an MV doesn't have an input voltage, so we will only show the output graph. As we can see, the MV is representing a square wave that is oscillating with a period of 0.38s with a amplitude of 3.5V.

MS1 Building Block 2: Diode based AND-gate

Schematic

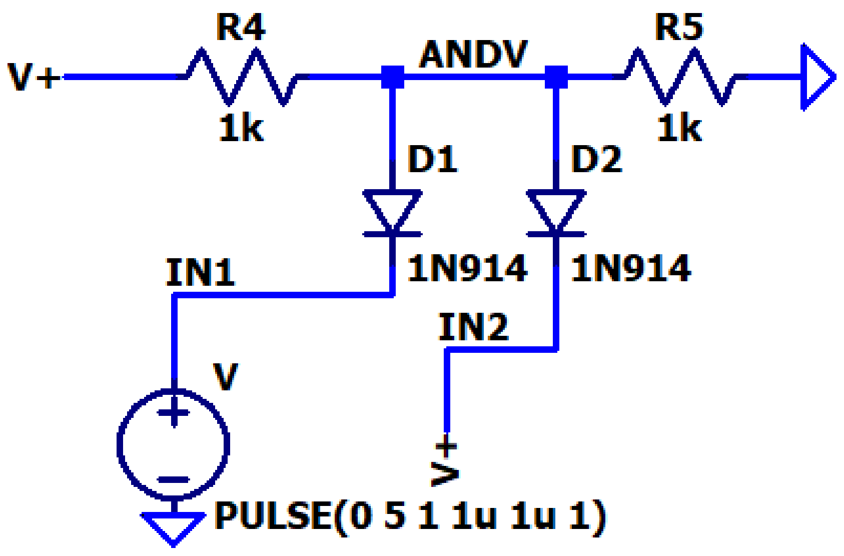


Figure 23: Diode Logic Gate Schematic

V+ is 5V, the voltage we would like to supply when both inputs are high. IN1 and IN2 indicate our inputs, and ANDV is the output of interest.

Design Equation

The AND logic essentially multiplies the two inputs together. Assume that 5V equates to binary input 1 and 0V equates to binary input 0. These are the only two inputs to consider. 1 * 1 = 1, while all other sets of inputs = 0. When the output is 1, it will turn on the LED for our output stage.

Discussion of Component Choices

The 1N914 diodes were used because they were readily available and met our requirements of acting like a one way switch with some voltage drop.

Input/Output Plot

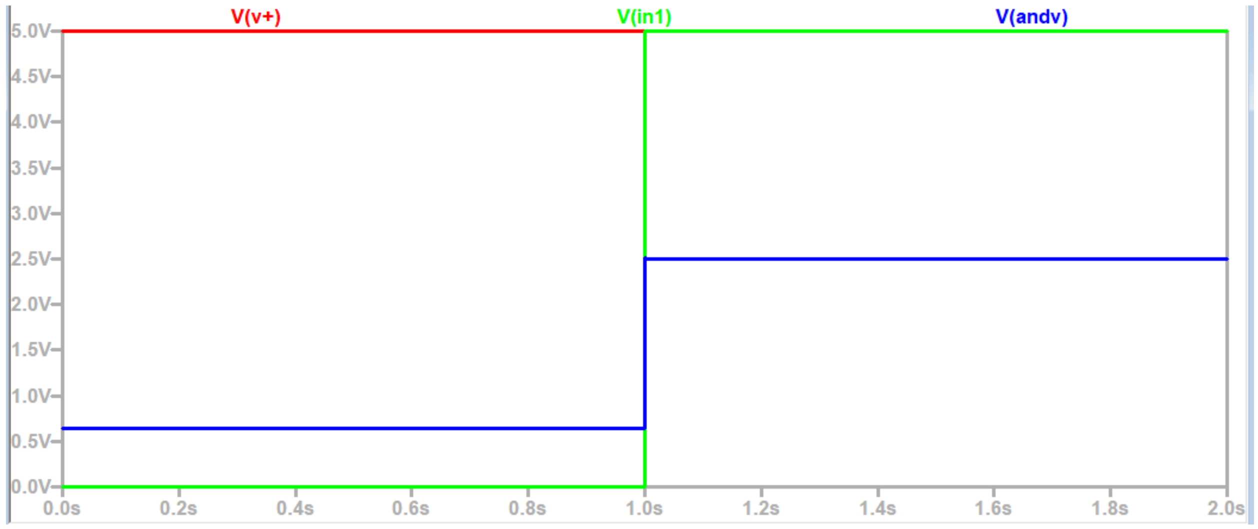


Figure 24: Input/Output of Diode Logic Gate

When both inputs V+ and IN1 are high(5V), output ANDV rises to its high state. Currently, the high state is lower due to a voltage divider. This is a limitation of LTSpice, as we need something to be the load for accurate measurements. The output of the ANDV will serve for the LED stage on whether it's on or off.

MS2 Building Block 1: CMOS based NAND-Gate

Schematic

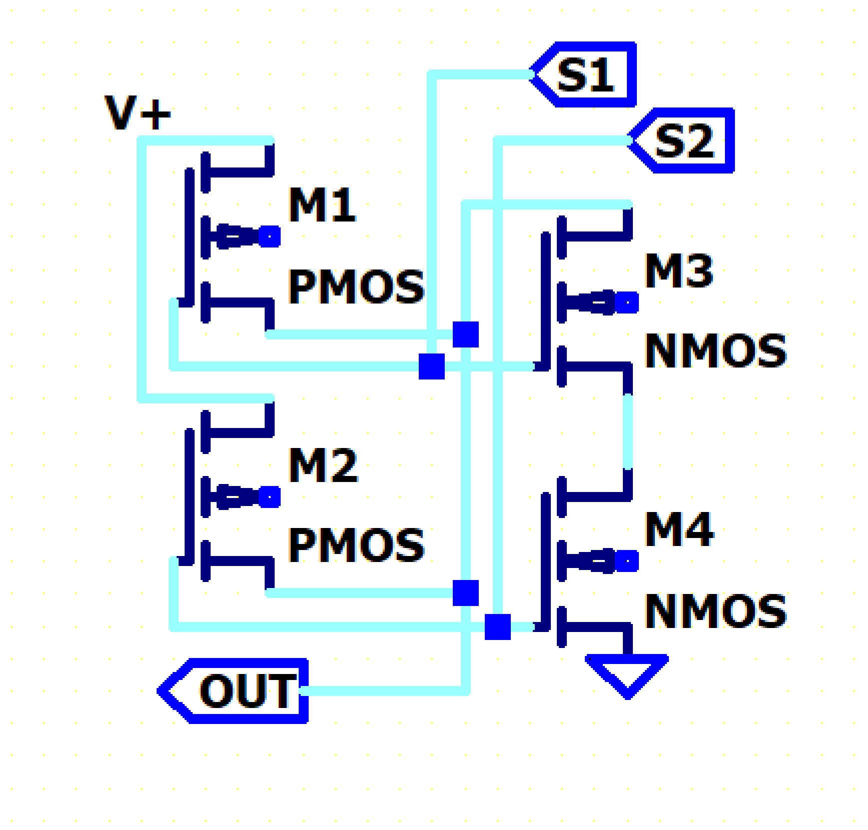


Figure 25: SPICE Model of a NAND gate

Design Equation

There is no explicit math in regards to a NAND gate. The only design principle that must be followed is the truth table, which is as below.

Table 1: NAND Gate input/output table

Input 1	Input 2	Output
LOW	LOW	HIGH
LOW	HIGH	HIGH
HIGH	LOW	HIGH
HIGH	HIGH	LOW

Discussion of Component Choices

The ALD1015 Dual N-Channel and Dual P-Channel MOSFET was chosen, as it was the only accessible PMOS. The only other MOSFET we had readily available was the 2N7002, and this is exclusively a NMOS.

Input/Output Plot

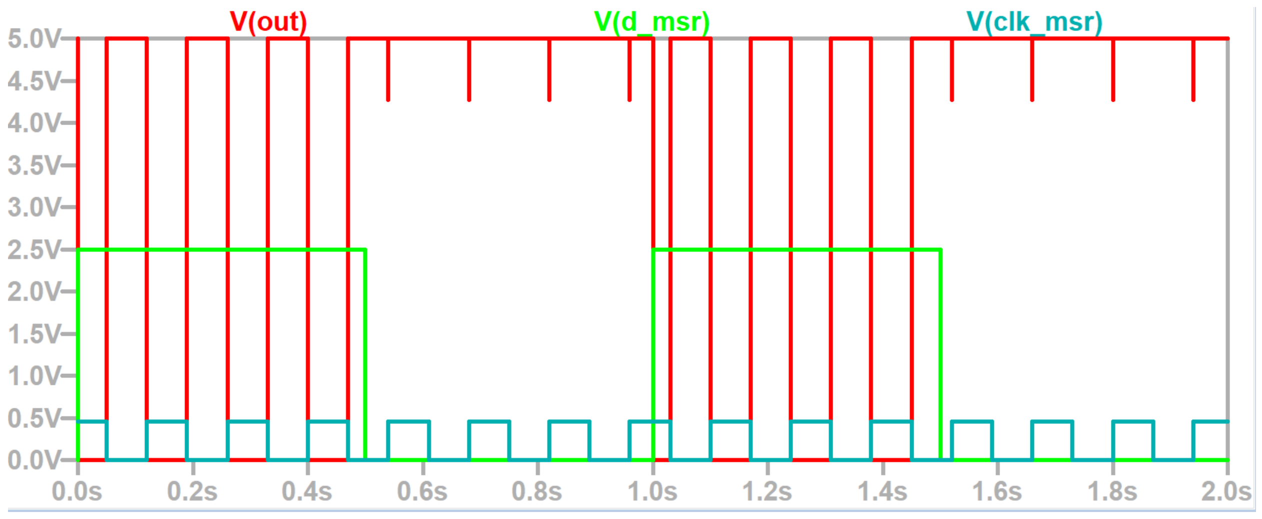


Figure 26: Input/Output of the CMOS NAND gate

Note how V(out) is only LOW when both inputs are HIGH, following the truth table of the NAND gate. (all inputs are 5V, just measured after a voltage divider for ease of viewing).

MS2 Building Block 2: Electronic Switch

Schematic

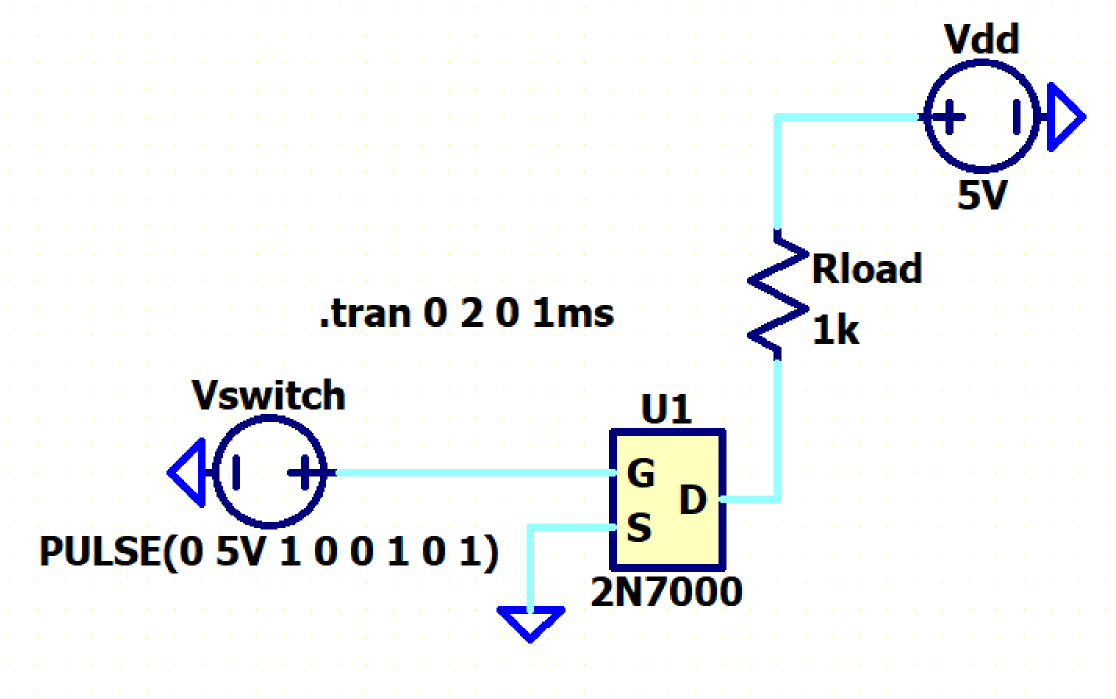


Figure 27: Electronic Switch with 2N7000 NMOS

Design Equation

If Vswitch > Vth, the switch is on. If not, the switch is off.

Discussion of Component Choices

The only consideration for this is Vth. As long as Vth is lower than our logic HIGH, the circuit should work correctly. As most MOSFETS we have access to have a Vth of around 0.7, we selected the 2N7000 for this, as it is the most accessible and fulfills this requirement.

Input/Output Plot

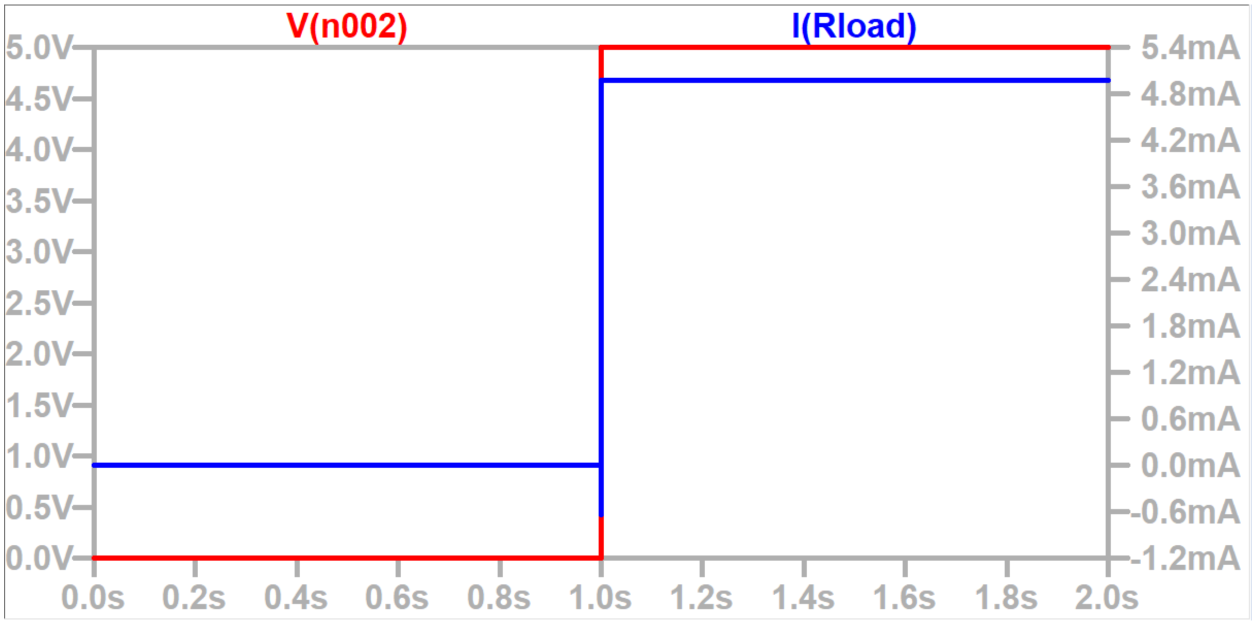


Figure 28: Input/Output of the Electric Switch

Note how as the switch gets toggled on, current begins passing through the load.

MS3 Building Block 1: Oscillator

Schematic

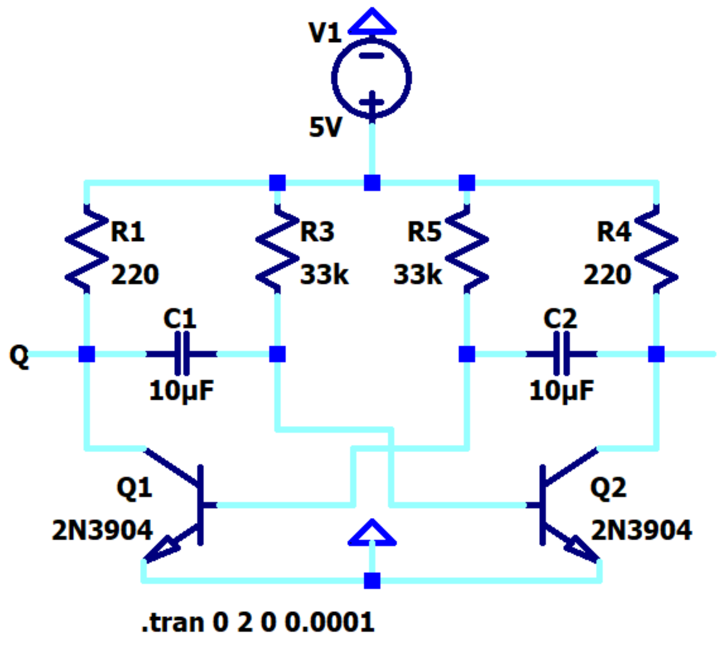


Figure 29: SPICE Model of BJT Astable Multivibrator Oscillator

Design Equation

From the Figure, we can see that R_3=R_5 and C_1=C_2. Which means the total length of time of the Multivibrators cycle is given below for a symmetrical output waveform.

$$f = \frac{1}{T} = \frac{1}{1.38*RC} \#(4)$$

$$T = 1.396 * RC \#(5)$$

Using the following equations, we can generate a continuous, oscillating square wave without requiring any external trigger.

Using equations # to find its characteristics of the square wave:

$$f = \frac{1}{1.38*(33000\Omega)(10*10^{-6}F)} = 2.2\ Hz$$

$$T = 1.396(33000)(10 * 10^{-6}) = 0.461s$$

Based on our frequency and the period, we can tell that our square wave will oscillate from 0V to 5V at a period of 0.461s.

Discussion of Component Choices

We first started out by choosing a value out for base resistor and collector resistor along with our capacitors. For the sake of convenience, we want our base resistor to be symmetrical as well as our capacitors, which we can have adjustable frequency by varying the values of the resistors and capacitors. As a result, since we want our target value of 2 Hz, so that the player has 1.5 second to react, we decided that R= 33k ohms while the capacitor is $10\mu F$ based on our available components in the kit.

Input/Output Plot

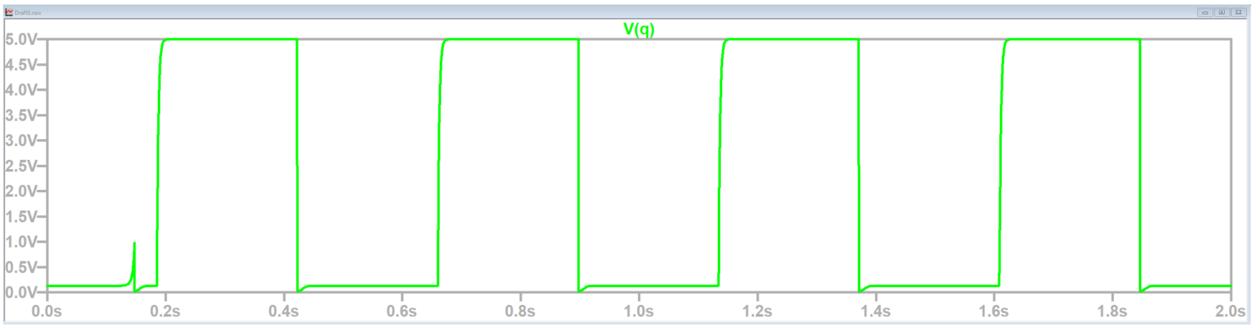


Figure 30: Output of Astable Multivibrator

Based on *Figure 30* and our analysis for the characteristics of the square wave, you can see that our calculation matches with the simulation. The circuit is oscillating from 0V to 5V at a frequency of 2.2Hz or at a period of 0.454s based on lTspice which is really close to our calculation.

MS3 Building Block 2: RTL Logic AND-Gate

Schematic

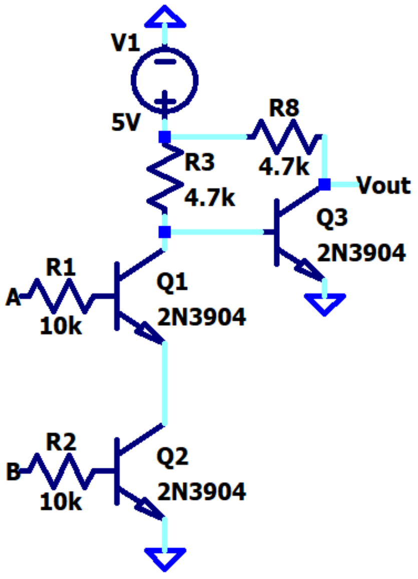


Figure 31: TTL AND-gate Schematic

Design Equation

There is no explicit math in regards to the AND gate, as the only design principle behind it is to follow the basic truth table logic of an AND gate. In written form, the AND logic essentially outputs a high value when both inputs are being detected, otherwise, the circuit will remain low.

Discussion of Component Choices

The same BJT 2N3904 was chosen due to its accessibility and readily available from the mercer lab. Since the TTL AND-gate input controls a multi-emitter transistor, each input is connected to one emitter of a single transistor where if any input is low, the transistor conducts, pulling the output low. A second transistor is used to invert the logic output where all inputs are HIGH, the transistor turns OFF, allowing the output to go HIGH.

Input/Output Plot

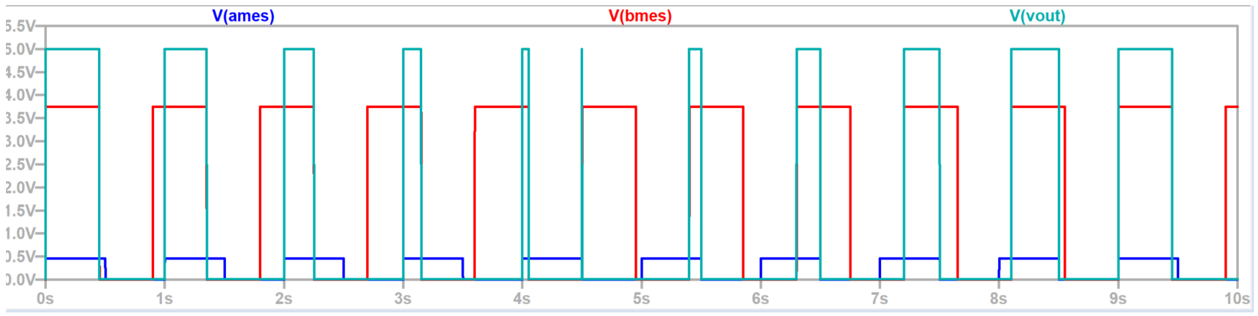


Figure 32: RTL AND Gate Ouptut

The above simulation output follows the truth table below:

Table 2: AND Gate truth table

A	B	OUT
0	0	0
0	1	0
1	0	0
1	1	1

Output Block: LED

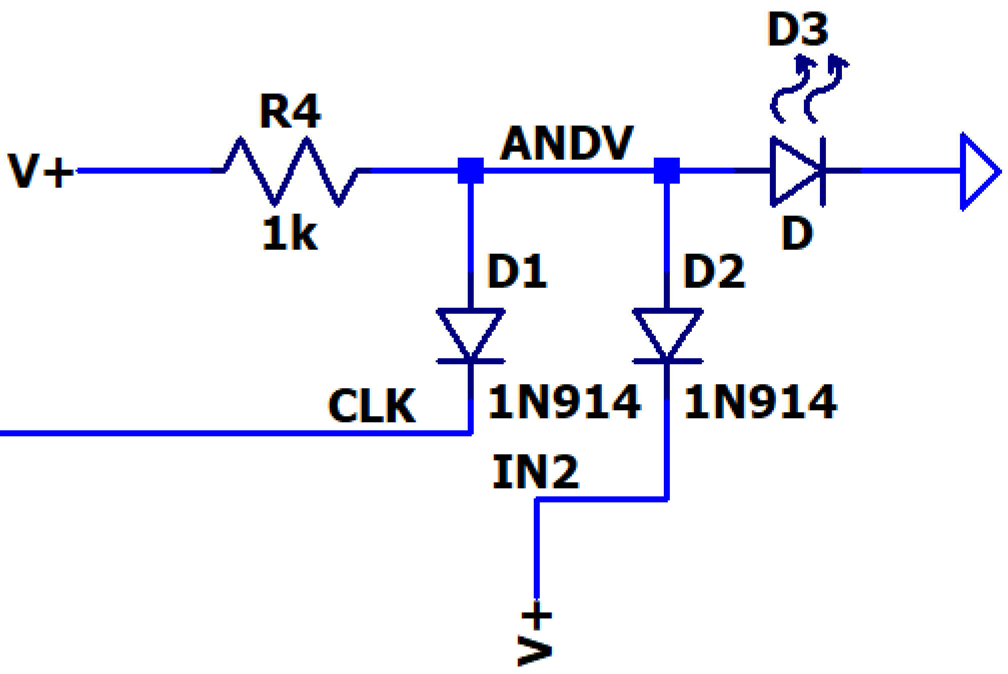


Figure 33: Example of a LED connected from an diode AND-gate

Our output is an LED. The LED is connected to the diode AND-gate as shown above. For our math and output of the LED, refer to the Diode based AND-gate *Figure 18* as it was explained there of how the logic worked. Based on the input of the diode AND-gate, when both of the desired inputs are true, then it would turn on the LED.

Integration and Optimization for MS1

Table 3: Inputs vs Outputs

Switch	Clock Signal	Output Voltage
ON	High	High
OFF	Low	Low
ON	Low	Low

OFF	High	Low

The first stage of our circuit is the slide switch that will connect to the diode AND-gate as one of its inputs. A side switch doesn't have anything too crazy about it other than having an on/off state which correlates to let through the signal pass through. The side switch has 3 pins where the first pin is the power, the last pin is ground, and the middle pin is the one that actually connects to the input of the diode AND-gate as previously discussed. In addition, the signal that goes out of the middle pin is a voltage.

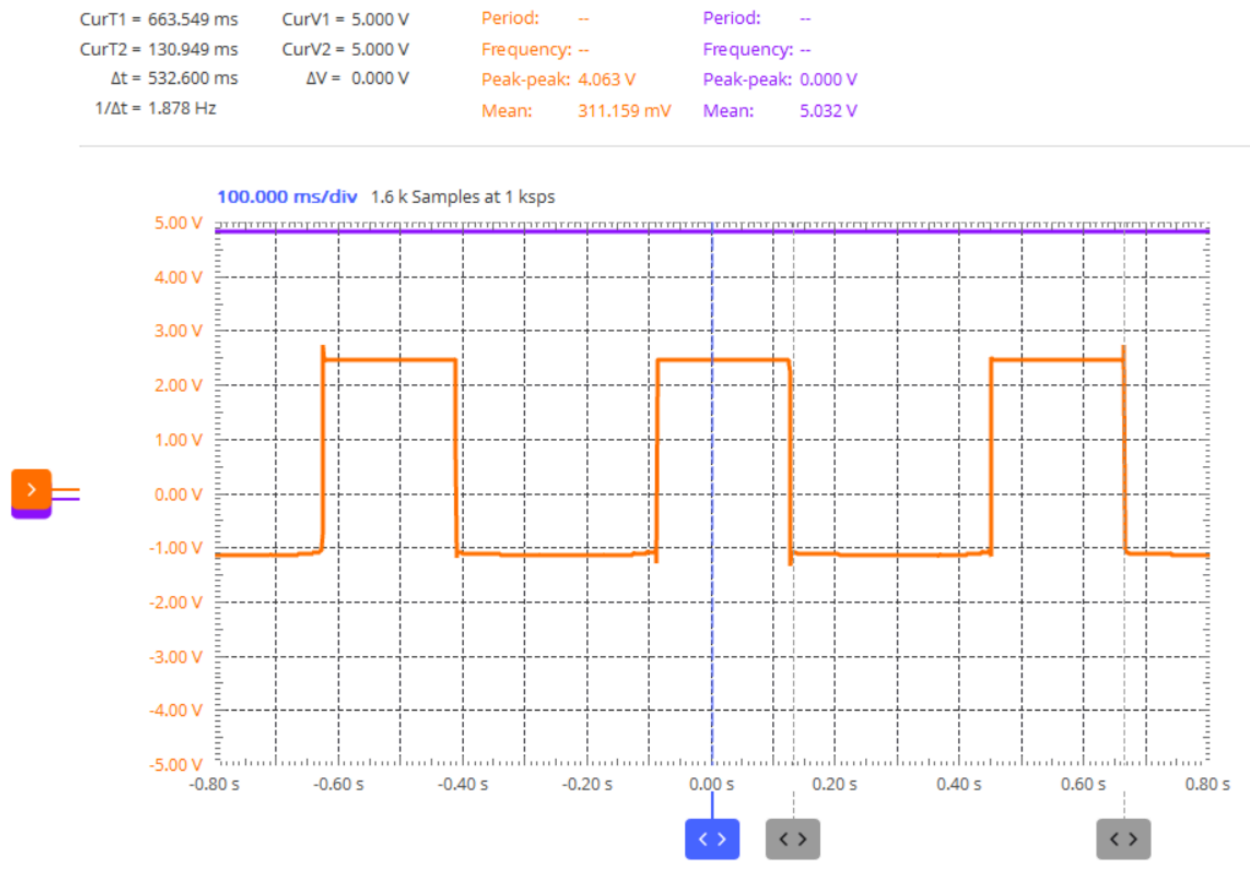
In the beginning stage, we originally planned to use a push button as our input stage. However, we later switched to a side-switch instead for the implementation. The reason is that the issue of the push button leads to the results to be inconsistent, which is why we made a decision for the demo and the project to be a side switch because the side-switch has a pin connection to ground, leading to results being consistent.

The second stage of our circuit is the op-amp multivibrator. As discussed in section "Astable Multivibrator," it will produce a voltage square wave signal (Vout) where the oscillation of the square wave can be used as a clock. For MS1, the MV will act as an individual source for the AND-gate to run for our game logic as mentioned before. It is designed with bounds of -5V and 5V in mind, but due to the limitations of the op-amp, it never quite reaches its saturation voltages. The 5V state is meant to be the logical high and the -5V state is meant to be logical low. With the op-amps imperfections and the load sapping some voltage, our current op-amp provides either 2.5V in its high state and -1V in its low state.

Originally, this was designed to oscillate between 0V and 5V, but such bounds were difficult to establish with the op-amps inability to reach its saturation points. Setting V_{sat-} to 0V was only able to pull the voltage down to 1V, which still provided enough voltage to constantly feed the AND gate with a logical high. To simplify our design for the sake of time, we went back to the reference point of 0V, which required equivalent + and - magnitudes for saturation. We used 5V again to satisfy our logical high requirement.

The third stage is the AND gate. To ensure that the AND gate was able to function well with both the slide switch and clock's outputs, we defined clear bounds as what would constitute a high voltage and a low voltage. Only a high voltage would let through enough power to light the LED that indicated jumping. This value was 0.7V, equivalent to the voltage drop of the diode. Lower than 0.7V, the diode wouldn't allow current to flow, effectively causing an open circuit. Above 0.7V, current flows and the LED is lit. The clock outputs 2.5V during its high, and the switch provides 5V when it's on, easily clearing the 0.7V threshold established.

Lastly, the final stage of our circuit is the LED from the output of the AND-gate configuration. Once the input logic is true from the MV's clock cycle on high and the side slide is on state, it will produce a Vout signal to the LED to indicate it is on, indicating that the event has been scheduled.



```
CurT1 = 663.549 ms     CurV1 = 5.000 V     Period:     --            Period:     --
CurT2 = 130.949 ms     CurV2 = 5.000 V     Frequency:  --            Frequency:  --
   Δt = 532.600 ms        ΔV =  0.000 V    Peak-peak:  4.063 V       Peak-peak:  0.000 V
 1/Δt = 1.878 Hz                           Mean:       311.159 mV    Mean:       5.032 V
```

Figure 34: Slide Switch On Input and Output

Figure 25 shows the input vs output of our entire circuit while the slide switch is on. The purple line represents our input, a slide switch in the ON position providing 5V, and the orange line represents the output, the two possible states of our clock. While the slide switch is on, the output is exactly the clock. Figure 26 shows the input vs output of the circuit while the slide switch is off. In this configuration, the yellow and purple lines overlap, as both of them read ground, the reference voltage of 0V.

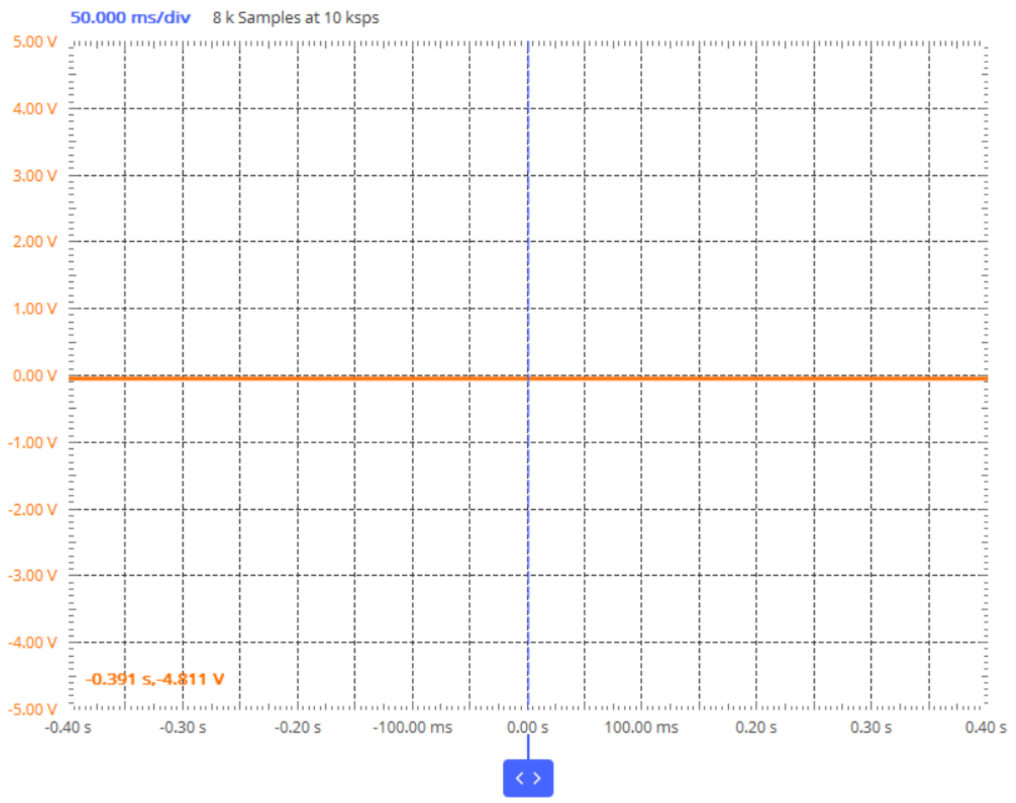


Figure 35: Slide Switch Off Input and Output

Integration and Optimization for MS2

Table 3: Inputs vs Outputs

D Switch	*Enable switch*	*Output Voltage*
ON	*OFF*	*PREVIOUS*
OFF	*OFF*	*PREVIOUS*

ON	*ON*	*HIGH*
OFF	*ON*	*LOW*

The first stage of our circuit is the two slide switches that are connected to the NAND gates as inputs. One of the slide switches (D switch) is configured to be our main input, while the other switch is the enable. This stage outputs true if both D and ENABLE are switched off, and off otherwise.

The second stage of our circuit is a second set of NAND gates, making a D Latch. The inputs to these NAND gates are the outputs of the previous NAND gates and its own output feedback as an input. This is represented by PREVIOUS in Table 3. The logic that the D Latch follows is shown in Table 3. As shown in the table, the D Latch will only change states when ENABLE is high. This creates a protected memory that is unalterable while we do calculations for the next game state. It ensures that state changes are invulnerable to noise and player inputs that may otherwise skew the state mid calculation.

We originally planned to use a T flip flop as the memory option of choice. A T flip flop worked the best because it would constantly flop every rising clock edge. This works perfectly for our desired 00, 01, 10, 11 binary states. However, a T flip flop proved too challenging to create in the alloted time. The created T flip flop was able to change states, but did not properly pull all the way to the designated voltage values of 0V and 5V. To complete the milestone in the alloted time, we opted to create a much simpler D Latch.

The third stage is an NMOS used as an Electronic Switch. This receives the signal out of the flip flop and uses it to open and close a voltage connection to an LED. The electronic switch will be used to automate the changing of inputs and outputs. By using a separate voltage source to power lossy components like resistors and LEDs that would otherwise drop voltage, we are able to keep the flip flop values at their logical high and low states of 0V and 5V. This ensures that our sequential logic will always have the same values to work with, whether it’s the first stage or the third. That way we can use the exact same layout for each of the stages, not needing to worry about minor differences in voltages or current.

The final stage is the LED stage which corresponds to the output of Table 2. The LED lights represent the binary state of the bit the flip flop represents. The bits from multiple flip flops will be combined and mapped to different combinations of LEDs lighting up, indicating important information such as whether or not the player is jumping and where obstacles are.

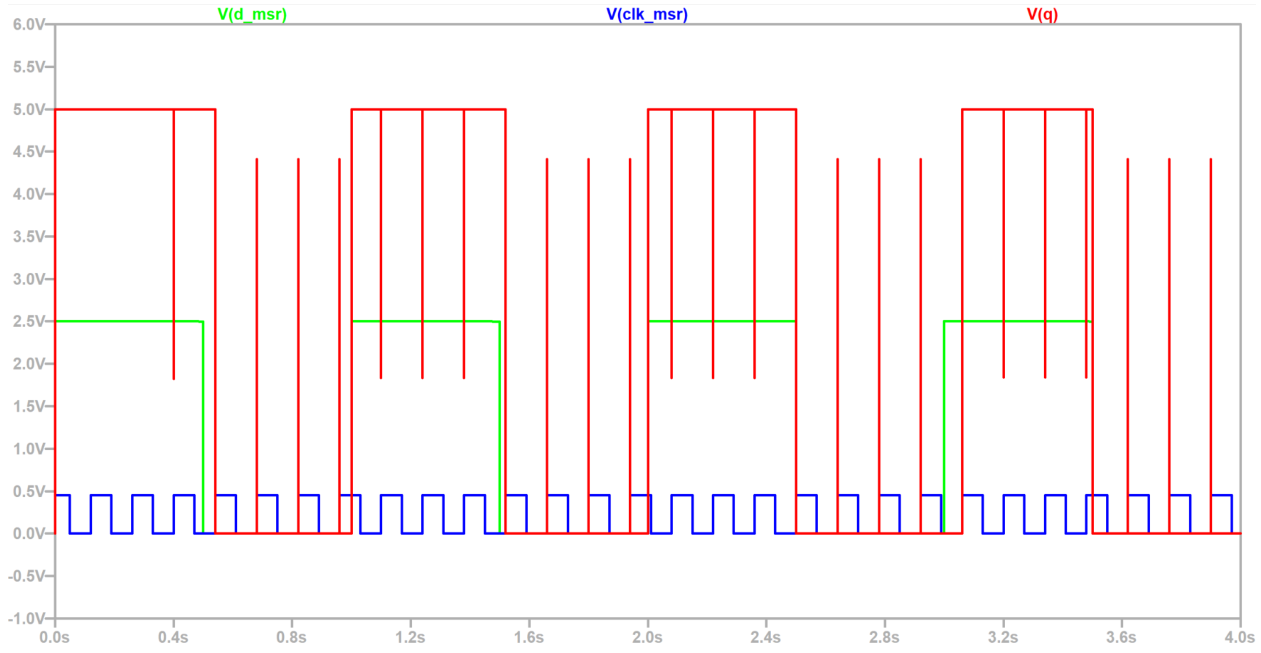


Figure 36: Input/Output of MS2 stage

As described in Table 3, V(q), our output, doesn't change until V(clk_msr) is high even when V(d_msr) is. All voltages are 5V high and 0V low, they are probed with voltage dividers for visual clarity. The red line V(q) is the output. There are some slight changes as some of the internal NAND gates that make up the latch change when clock rises, but these changes are imperceptible on our normal circuit.

Integration and Optimization for MS3

The circuit begins with the clock. It bounces between a 5V high and a 0V low roughly every half a second, which is in line with our 2 Hz target. Alongside this "clock," there's a button that you can push to indicate to the game that you wish to jump. Both the clock and the button feed into our memory unit, which is an Arduino. We intended for this to be a series of flip flops, but when we realized the number of flip flops we needed, we quickly gave up on the idea. We originally planned a flip flop per LED, meaning 3 flip flops, but we turned that into 2 flip flops after some truth table tinkering. The equations we ended up using can be seen in Table below.

Table 4: LIght Sequence

A	B	LED NO
0	0	ALL LEDs OFF
0	1	1
1	0	2
1	1	3

Two flip flops seemed doable, but then we remembered we needed memory to know the previous state of the LEDs to know where the player stands in the game state diagram. That quickly ballooned it back up to 6 flip flops, way too many considering each flip flop took an entire breadboard. That's where the Arduino came in. The Arduino allowed us to store the flip flop states as variables all in one convenient little board. It also served to provide signals to indicate the previous state to our decision making circuit. This decision making circuit was one of our RTL logic gates configured as an AND gate, where it and'ed whether or not you hit the jump button with if the 3rd light was on to decide whether or not you lost.

We explored RTL logic gates using BJTs in order to ensure that we gave all the possible logic gate varieties a try. In MS1, we explored Diode logic, which we promptly replaced with CMOS logic in MS2 due to its extreme lossiness. Diode logic is old, found in the early days of mainframe vacuum tube based computing. The CMOS logic we explored in MS2 is the same used in the most cutting edge of computing technology today. They are able to be scaled incredibly small and packaged together into tiny chips with very little power draw. RTL logic bridges that gap, being smaller and more efficient than diode logic as it can go to the full reference voltages but still not as efficient as CMOS logic with its zero power draw unless switching.

RTL logic worked like a charm in small quantities, but we soon discovered its main downfall: scalability. Because it used resistors, it lost voltage to heat. It needed constant level shifts to maintain its full signal strength throughout the logic sequence. A sequence of RTL logic gates also triggered the M1K's infamous beeping, which signaled it was drawing too much current from it. And because BJTs were super sensitive to small differences in current, attaching a load was a nightmare. Even just placing in an LED would cause multimeters to read 0V output from the RTL gates, despite there somehow being enough power to light up the LED.

Due to the issues with RTL logic, we ended up going back to CMOS logic in our final integration.

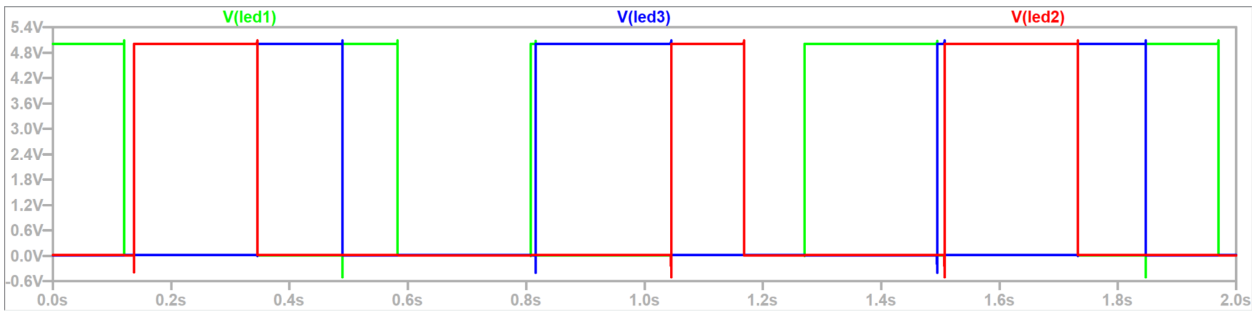


Figure 37: Whole Circuit Output

The output for our circuit is *Figure 37.* It shows how the sequence of lights LED1, LED2, and LED3 trigger one after the other, although the ordering and timing may be off due to LTSpices lack of a usable memory unit for analog devices

Operating Conditions MS1

Limitation 1:

Flipping the switch too close to the clock's falling edge will sometimes fail to trigger the jump LED despite the clock high LED indicating there is still time to do so. This is because the voltage is not entirely stable during the falling edge. The voltage from the astable multivibrator is falling at this time. The diode AND gate requires a high signal from the op-amp, while the clock high LED only requires some current, so the clock high LED can be lit without providing enough voltage for the AND gate to read logical high.

Limitation 2:

We can only have up to 3 stages of sequential logic with our current design. This is because diodes drop voltage. We designated 3.3V as our logical high to mimic that of popular microcontrollers like the STM32. With the ~0.7V drop that we observed, our 5V inputs can only be dropped 3 times before falling too low for our designated logical high. As we implement more complex logic into our circuit, we need to either rethink our logic gates or find a way to restore our voltage after every logic gate.

Trade Offs:

In the design of our op-amp multivibrator, we had to choose between precision and simplicity of design. A multivibrator with a higher frequency would allow us to run logic more often, but would also require additional circuitry to ensure that we're only doing things every 2 ticks, 3 ticks, etc. to keep the timing consistent. As this is the first time we are building a circuit that requires such tight synchronization, we decided that it's more important to focus on keeping things simple and making sure they work than adding all the features we want.

Improvements:

The first potential improvement is to the astable multivibrator. It should have a higher frequency. That way, we can implement things like making the game harder as time progresses by reducing the amount of time you have to react and jump. As of right now, each tick of the clock correlates to one change of state. But that doesn't have to be the case. We can program it so that every 3 ticks changes the state, then 2, then 1 as you progress further and further into the game so the game continuously gets more difficult as time progresses. If time allows, we will try to implement this in future milestones.

The second improvement is to try to replace the switch with a button. A button is much more intuitive than a switch. Currently, there are issues with the button as an unpressed button creates an open circuit. We need to ensure that when the button is not pressed, the reading is ground for a perfect 0V. With an open circuit, floating voltage can cause unintended effects as it's neither a true logical low or logical high. We will continue to explore ways to implement this as we work through future milestones.

Operating Conditions MS2

Limitation 1:

The circuit is very vulnerable to noise. If there is a sudden spike or drop of voltage at the very last second while the enable is high, the circuit will remain in that glitched state for the entire low period. While the inputs may change for the entire duration that enable is high, only the voltage when enable is low will be stored as memory. Noise from the power supply won't be a very likely thing, but noise from the feedback loop will be. If you try to change the states right before enable goes low and the state change doesn't finish, you'll be stuck at a half-state.

Limitation 2:

The circuit is slow. This makes the flipping of states unsteady for a split second, not noticeable on our output LED, but we have noticed it in our oscilloscope measurements. This can cause issues with our feedback loop, which relies on the other NAND gate's state to determine its own. While it has no effect now, if we were to increase the switching frequency we may run into issues with stability and making sure our logic properly pulls up and down to the 5V and 0V designated as logic highs and logic lows. To combat this, the clock that dictates the change in game states should be kept at a very low frequency to allow for the values to fully settle before changing again.

Trade Offs:

To maintain simplicity and fit our design onto one singular breadboard, we went with a D flip flop instead of a T flip flop. A T flip flop would better serve our purposes, as it always toggles when enabled. We want one flip flop that always triggers, and another that only triggers if the first flip flop was 1 the previous state. That way, we can easily set up 00, 01, 10, 11, reset as our values, which maps perfectly into the 4 possible states of our obstacles. A D flip flop can accomplish the same thing, but we need something to alter the D values every other clock cycle because it doesn't do it by itself.

Improvements:

Due to time constraints, we made a latch instead of a flip flop. A very simple improvement would be to upgrade the latch into a flip flop by adding a slave latch that has the master latch's output Q as its D. This simple change will help synchronizing things a lot by locking the states for longer periods of time. Right now, the state is only locked while enable is low. With a flip flop, the state is locked for all times except when enable is triggered.

A second improvement would be to add capacitors to the supply and input voltages to help against sudden voltage spikes/drops. This way, we can guarantee consistent values without micro-fluctuations that could potentially chain together to cause unintended effects. Due to the sheer size, it’s very important to stop problems that are too small to care about otherwise as they could really build up after 10-20 sequences.

Operating Conditions MS3

Limitation 1:

There is a chance for the analog reading to “float.” A floating voltage is a voltage that isn’t fully high or fully low. This occurs because the button isn’t grounded when it’s not pressed. Normally, floating voltages hang low, but sometimes if other parts of the system are really noisy, the floating values can spike into the territory designated for high values. We attempted to protect it by adding a resistor to pull down the voltage fully, but that means now the button voltage is a bit lower because it’s split with a resistor as well as the load. Because of the Arduino’s capability to dial in the thresholds on the field, we were ok, but we can run into issues from it when we start to exceed the amount that can fit onto an Arduino

Limitation 2:

The components in this milestone (BJTs, LEDs, resistors) all consume too much power. When loading the entire system, certain parts may not work as intended due to fluctuations in voltage and/or current as the M2K struggles to provide the demanded power. The current MOSFet as switch solution where we wired a separate power source to some higher demand components introduces small bits of delay that can add up as the game goes on. There have been instances where the LED’s state doesn’t line up with the actual states of the variables due to this delay.

Trade Offs:

In order to keep our project at a doable scale, we decided to pivot away from using flip flops and instead chose to use an Arduino. Flip flops would provide an entirely analog experience without floating values and would be much more in sync with the clock

Improvements:

The first improvement would be adding feedback to the game. There is no feedback to the game. You don't know if the game just ended by accident or if you actually jumped over the obstacle because the obstacle disappears in front of you either way. Making the player a BiLED with a different death color or having LEDs behind the player so that the player can see the obstacle roll past them would help greatly with increasing the clarity of our game. It was not implemented due to the increased difficulty it would bring. With a BiLED, we would have to find a way to control both side's voltages. With extra obstacle position LEDs, we would have to add even more memory variables.

The second improvement would be to use a lower voltage. 5V was chosen because it was the maximum value the M2K boards could provide and we wanted to have as much power as we might need. However, we never needed that much power. In the future, we could redo this with as little as 3V, just enough to tip the MOSFETs we're using. The lower power draw means we wouldn't run into power issues and can access a larger suite of power hungrier options for our build.

ABET DESIGN CONSIDERATIONS

Because we are designing a simple game, we didn't consider its impact on the public in our design choices. The Dinosaur Game project was intended to help us explore our curiosity in a field, not attempt to change the world for the better. The design process placed a heavy emphasis on power efficiency, as the sources we had easy access to simply didn't provide much power. In that aspect, our game is environmentally conscious, choosing the least power hungry parts while still providing a solid experience. Our game was also designed safely, so that it won't injure the player. But in the grand scheme of things, those considerations have no effect on society.

References

[1]“Dinosaur Game,” Wikipedia, Sep. 01, 2021. https://en.wikipedia.org/wiki/Dinosaur_Game

[2]“Op-amp Multivibrator or Op-amp Astable Multivibrator,” Basic Electronics Tutorials, Aug. 24, 2013. https://www.electronics-tutorials.ws/opamp/op-amp-multivibrator.html

[3]”Wide bandwidth single JFET operational amplifiers,” ST, Apr. 2008. https://www.st.com/resource/en/datasheet/lf351.pdf

[4]GeeksforGeeks, “Sample and Hold Circuit,” GeeksforGeeks, Feb. 07, 2024. https://www.geeksforgeeks.org/sample-and-hold-circuit/

[5]“Logic Gates in Digital Electronics Complete Guide Electronic Clinic,” Electronic Clinic, Mar. 17, 2020. https://www.electroniclinic.com/logic-gates-in-digital-electronics-complete-guide/

[6]“Small Signal Fast Switching Diodes ADDITIONAL RESOURCES MECHANICAL DATA.” Available: https://www.vishay.com/docs/85622/1n914.pdf

[7]“N-Channel Enhancement-Mode Vertical DMOS FET,” MICROCHIP, 2021. https://ww1.microchip.com/downloads/en/devicedoc/2n7000-n-channel-enhancement-mode-vertical-dmos-fet-data-sheet-20005695a.pdf

[8]“ALD1105 DUAL N-CHANNEL AND DUAL P-CHANNEL MATCHED MOSFET PAIR.” Available: https://www.aldinc.com/pdf/ALD1105.pdf

[9]admin, “CMOS Logic Gates Explained,” ALL ABOUT ELECTRONICS, Apr. 14, 2023. https://www.allaboutelectronics.org/cmos-logic-gates-explained/

[10]O. M. Urias, “The D Latch (Quickstart Tutorial),” Build Electronic Circuits, Dec. 13, 2022. https://www.build-electronic-circuits.com/d-latch/

[11]“2N3903, 2N3904,” ONSEMI, August, 2021. https://www.onsemi.com/download/data-sheet/pdf/2n3903-d.pdf

[12]“Logic AND Gate,” Electronics-Lab.com.
https://www.electronics-lab.com/article/logic-and-gate/

[13]Arduino, “UNO R3 | Arduino Documentation,” docs.arduino.cc, 2024.
https://docs.arduino.cc/hardware/uno-rev3/

[14]Electronics Tutorials, “Astable Multivibrator and Astable Oscillator Circuit,” Basic Electronics Tutorials, Sep. 05, 2013. https://www.electronics-tutorials.ws/waveforms/astable.html

[15]“BJT Astable Multivibrators,” learnabout-electronics.org.
https://learnabout-electronics.org/Oscillators/osc41.php

Appendix

Code used:

```
//global variables

//ensures clock is only read once per up/down cycle

int CLOCK_HI = 0;

//lockout for jump

int jumped = 0;

//jump variable

int jump = 0;

//which LED was on in the previous state?

int LED0 = 0;

int LED1 = 0;

int LED2 = 0;

//trigger LED sequence

int LED = 0;

//game state

int game = 1;

//test

int enable = 0;

//score tracking

int score = 0;


void setup() {

  // put your setup code here, to run once:


  //pin declaration

  //LIGHT0
  pinMode(2, OUTPUT);

  //LIGHT1

  pinMode(3, OUTPUT);

  //LIGHT2

  pinMode(4, OUTPUT);

  //JUMP

  pinMode(5, OUTPUT);


  Serial.begin(9600);

}


void loop() {

  // put your main code here, to run repeatedly:

  if(game == 1) {

    //check for a high voltage value

    int clockVolt = analogRead(A0);

    int jumpVolt = analogRead(A1);

    //Serial.println(jumpVolt);

    //int testVolt = analogRead(A2);

    //Serial.println(testVolt);


    /*if(testVolt > 800) {

      enable = 1;

    } */


    if((jumpVolt > 800) & (jumped == 0)) {

      jump = 1; //buffer jump for next cycle
    }

    if((clockVolt > 800) & (CLOCK_HI == 0)) {
      //Serial.println("High");
      CLOCK_HI = 1;
      onClockRise();
    }

    if((clockVolt < 300) & (CLOCK_HI == 1)) {
      //Serial.println("Low");
      CLOCK_HI = 0;
      onClockFall();
    }
  }
}

void onClockRise() {
  Serial.println("HIGH");
  if(jumped == 1) {
    //no longer safe
    jump = 0;
    digitalWrite(5, LOW);
  }
  if(jump == 1) {
    //safe for a single cycle, locked out for 2
    jumped = 2;
    digitalWrite(5, HIGH);
```

```
}
//check LED2, then LED1, then LED0, then LED
if(LED2 == 1) { //code for when rightmost is on
  LED2 = 0;
  //turn off LED
  digitalWrite(4, LOW);
  if(jump == 0) {
    game = 0;
    Serial.println("You Lose!");
  } else {
    score = score + 1;
    Serial.print("SCORE: ");
    Serial.println(score);
  }
}
if(LED1 == 1) { //code for when middle is on
  LED1 = 0;
  LED2 = 1;
  //change LED from middle to left
  digitalWrite(3, LOW);
  digitalWrite(4, HIGH);
}
if(LED0 == 1) { //code for when left is on
  LED0 = 0;
  LED1 = 1;
  //change LED from right to middle
  digitalWrite(2, LOW);
```

```
    digitalWrite(3, HIGH);
  }
  if(LED == 8) { //code for triggering left
    Serial.println("LIGHTED UP!");
    LED = 0;
    LED0 = 1;
    digitalWrite(2, HIGH);
  }
  /* if(enable) { //automatically trigger obstacle spawn
    enable = 0;
    LED0 = 1;
    digitalWrite(2, HIGH);
  } */

}

void onClockFall() {
  //cycle 1 is cycle jump is buffered for, cycle 2 is cycle after. locked
  if(jumped > 0) {
    jumped = jumped - 1;
  }
  LED = LED + 1;
}
```