# Introduction

This project aims to design and build an automatic lamp-post and counting system for a slot car race track set as shown in Figure 1.

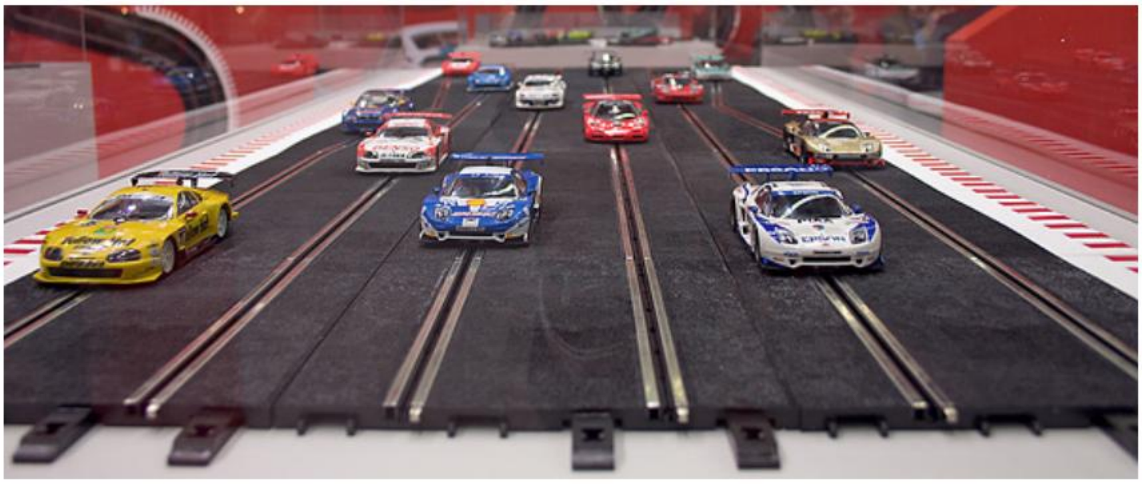


*Figure 1: Slot cars and track (Thomas Mielke / CC-BY-2.0-DE)*

The system works by moving a slot-car in front of a proximity sensor, which reflects IR LED (transmitter) beam to an IR photodiode (receiver). If the IR signal is strong enough, the lamp-post driver will activate the lamp-post LEDs for as long as the object is near the proximity sensor. The lamp-post driver signal is sent into the MCU via a digital interface circuit. This circuit counts the number of items that pass through the sensors and displays the appropriate figure on the output display.

This project is to be implemented across three sections; the first section *Analogue Electronics* aims to design the proximity sensor that drives an automatic lamp-post circuit, the following section *Microcontroller and Output Display* involves connecting a Raspberry Pi Pico microcontroller board to a dual-digit, 7- segment output display, and then programming it to show any of the numbers between 0 and 99. The final section *Interface Electronics* comprises a higher-level design of the interface that connects the electronics from the preceding sections to build the complete sensor system.

# Analogue Electronics

## 14/03/2024

## Introduction & Aims

The primary objective for this experiment is to design a proximity sensor to drive an automatic lamp-post circuit as shown in Figure 2. It consists of two main parts: a transmitter, and a receiver. The transmitter circuit comprises an infrared (IR) LED emitting a radiation of 890 nm wavelength when powered. The receiver is a photodiode that generates a current when it's exposed to IR radiation with a wavelength between 870 to 950 nm. Both devices are positioned so that when an object is in close proximity, the radiation from the IR LED is reflected back to the photodiode, which generates a current that triggers the lamp-post driver.

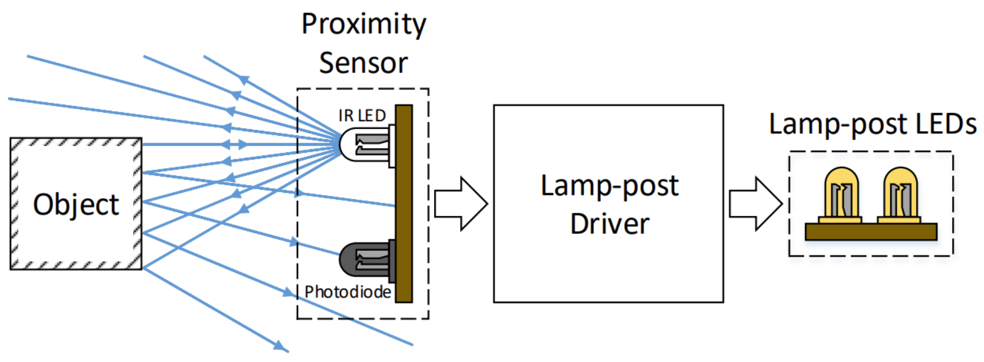


*Figure 2: Auto-lamppost sub-systems overview.*

The procedure followed here is to start by constructing the main building blocks of the lamp-post sub-system in the *theoretical design* sub-section; the proximity sensor circuit, the lamp-post driver, and the lamp-post Light Emitting Diodes (LEDs) circuit. Then choosing any needed experimental values for the components in the *Experimental Procedure & Results* sub-section. Finally, everything is assembled at the end to create the complete sub-system.

## Theoretical Design

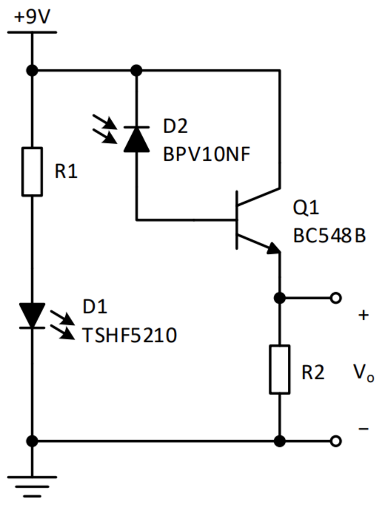


*Figure 3: Proximity Sensor Circuit Schematic*

The proximity sensor circuit schematic is depicted in Figure 3, the transmitter circuit on the left comprises a resistor $R_1$ connected in series with a TSHF5210 LED [1] with a radiation wavelength of 890 nm. The receiver part on the right consists of a BPV10NF IR LED [2], a resistor $R_2$, and a BC548B [3] bipolar junction transistor (BJT) of type NPN. This NPN BJT amplifies the current fed to its base by the photodiode upon being exposed to the IR radiation coming from the transmitter part. The current amplification follows this equation:

$$I_C = \beta I_B \qquad (1)$$

Where $I_C$ = Collector current (A)

$I_B$ = Base current (A)

β = DC current gain

The values of $R_1$ and $R_2$ are to be found experimentally in the next subsection. However, before starting to tweak the value of $R_1$, it should be ensured that the maximum forward current of the TSHF5210 LED isn't exceeded. For this, a safety resistor $R$ is introduced, which is connected in series with a variable resistor $R_{v1}$, as shown in Figure 4. Note that $R_1 = R + R_{v1}$.

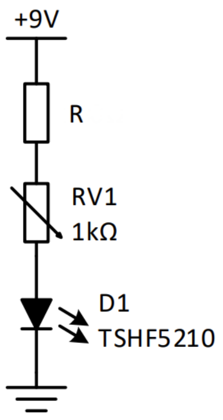


*Figure 4: Transmitter circuit with safety resistor R*

The minimum value for R that ensures minimum current is calculated through the following equation:

$$R_{min} = \frac{V_{DD} - v_{max}}{i_{max}} \qquad (2)$$

Where $V_{DD}$ = power supply voltage (V)

$v_{max}$ = Diode's maximum forward voltage (V)

$i_{max}$ = Diode's maximum forward current (A)

The maximum forward current of the TSHF5210 LED used in the transmitter circuit (as from the datasheet [1] ) is 0.1 A, and has a forward voltage of 1.4 V; the power supply voltage is 9 V. Substituting these values in equation 2 yields $R_{min}$ = 76 Ω.

The next part is to build the lamp-post driver system, which is a threshold comparator circuit constructed using an operational amplifier (op-amp). When configured without feedback or with positive feedback (Figure 5a), the op-amp compares the voltages at its non-inverting ($V^+$) and inverting ($V^-$) inputs to determine the output ($V_o$). It saturates to the positive power rail ($V_H$) If $V^+ > V^-$, and to the negative power rail ($V_L$) when $V^+ < V^-$. This behavior is illustrated in Figure 5b.

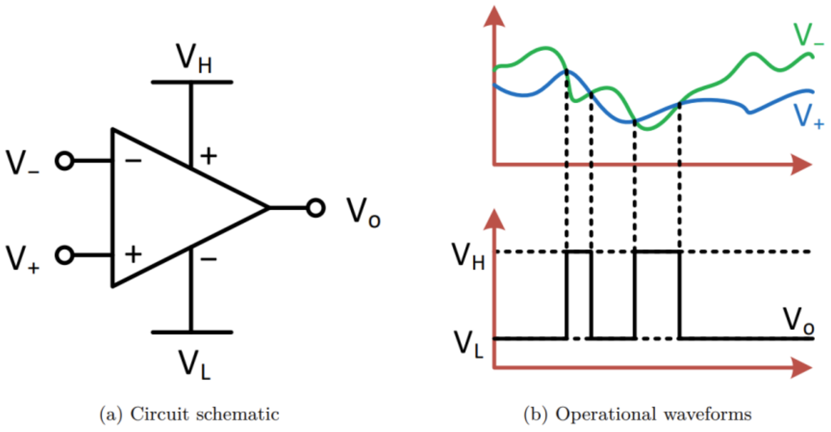


*Figure 5: Operational-Amplifier Comparator Operation*

The threshold comparator configuration involves connecting a potentiometer to the op-amp. The potentiometer serves as a voltage divider, setting the circuit's threshold voltage, which is fed to its inverting input, as depicted in Figure 6.

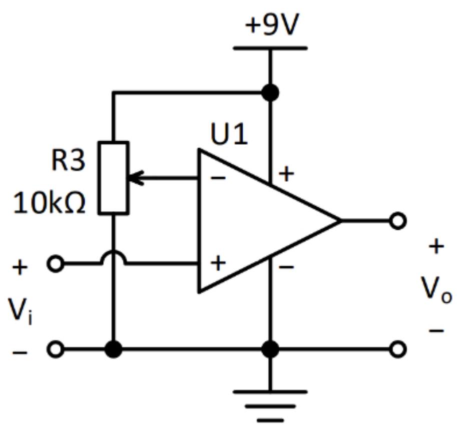


*Figure 6: Threshold comparator.*

Adjusting the potentiometer's configuration changes the value of the threshold voltage according to the following equation:

$$V_{th} = kV_{DD} \qquad (3)$$

Where $0 < k < 1$ is a variable represents the potentiometer configuration, as shown in Figure 7 below.

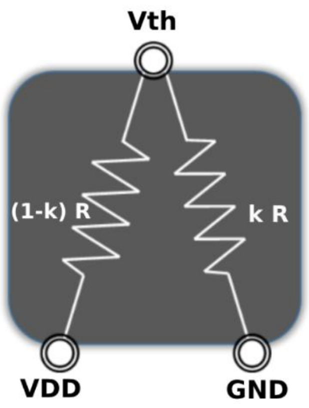


*Figure 7: potentiometer as a voltage divider.*

The final part is to construct the lamp-post circuit (Figure 8), which comprises two C503D LEDs [5] and a resistor. Similar to the transmitter's circuit, the resistor's purpose is to limit the current flowing through the LEDs and ensure that it stays below the maximum value specified in the datasheet.

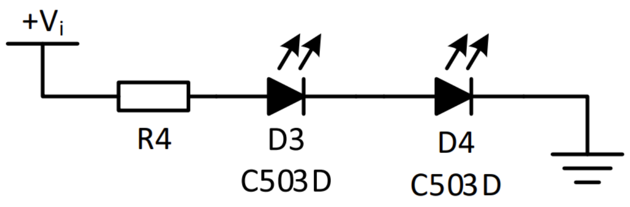


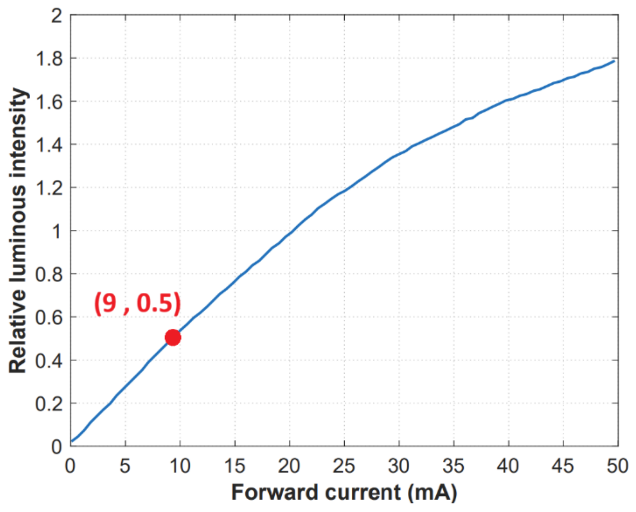
*Figure 8: Lamp-post LEDs circuit.*

The value for $R_4$ is chosen based on the desired LEDs relative luminous intensity, once this is determined, then the value for the forward current $i_f$ is found from the Relative Luminous intensity vs

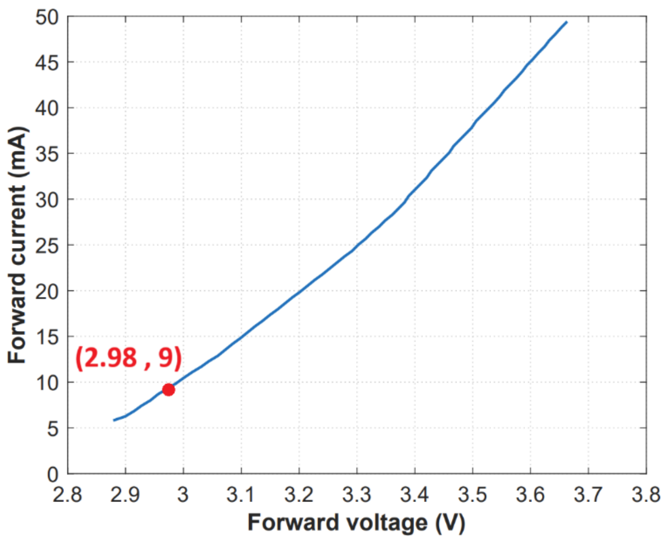
Forward Current graph from the LED datasheet. The same procedure is then followed to determine the forward voltage $v_f$ for each of the LEDs from the Forward Current vs Forward Voltage graph. Then, the values of $i_f$ and $v_f$ are substituted in equation 4 below to determine $R_4$.

$$R_4 = \frac{V_{DD} - 2 \times v_f}{i_f} \qquad (4)$$

Figures 9a & 9b below demonstrate the preceding procedure for finding $i_f$ and $v_f$ for the circuit in Figure 8 operating with a relative intensity of 0.5, which was selected to carry forward with this experiment.



*a) Relative luminous intensity vs Forward current plot*



*b) Forward current vs Forward voltage plot*

*Figure 9: C503D LED properties with relevant points marked when operating at a luminous intensity of 0.5.*

The previous plots show values of 9 mA, 2.95 V for $i_f$ and $v_f$ respectively, needed to operate at 0.5 intensity, plugging these values in equation 3 yields $R_4$ = 344 Ω.

## Experimental Procedure & Results

## 21/03/2024

Before assembling the preceding building blocks of the complete sub-system, suitable values for resistors $R_1$, $R_2$, and the potentiometer variable $k$ need to be determined experimentally.

For resistors $R_1$ and $R_2$, their values need to result in the highest voltage change across R2 in the case when there is an object in close proximity, and when it is moved away. Recall that $R_1 = R + R_{v1}$. Minimum value for $R$ of 76 Ω was derived previously, a 100 Ω $\pm$ 5% resistor was employed in this experiment instead for convenience.

Different combinations of $R_{v1}$ and $R_2$ values through 1k and 100k potentiometers respectively were tested. The following resistance nominal values for $R_{v1}$ and $R_2$ where chosen:

$R_{v1}$ (Ω): 0, 250, 500, 750, 1000.

$R_2$ (kΩ): 20, 40, 60, 80, 100.

Each value for $R_{v1}$ was tested with the five values of $R_2$, resulting in 25 different combinations. For every combination, the voltage across $R_2$ was recorded before and after placing a flat object at a distance of 10 cm. The difference between the two measurements was calculated and then used to produce five plots of this quantity vs $R_2$ for each of the $R_{v1}$ values, as depicted in Figure 10.

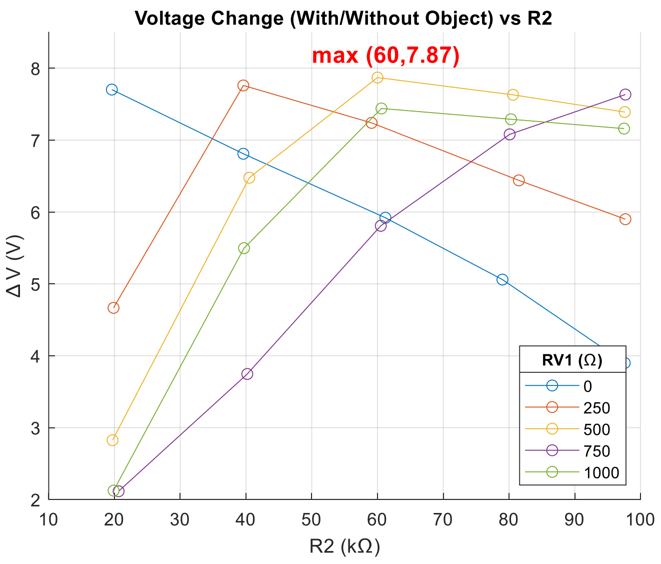


*Figure 10: change in voltage vs R2 graph in case when object is in range and when not, for different values of R1.*

The previous graph shows that the highest voltage change occurs at values ($R_{v1}$, $R_2$) = (500 Ω, 60 kΩ). Note that $R_1$ = 100 + 500 = 600 Ω, which is what will be employed for the rest of the experiment.

For this combination, the voltage recorded when an object was brought to a distance of 10 cm from the sensor was 8.63 V, and when moved away 0.76 V (hence the change = 7.87). A suitable comparator's threshold voltage here was chosen to be the mid-point of the two values, that is:

$V_{th} = \frac{V_{w_{obj}} + V_{wout_{obj}}}{2} = \frac{8.63+0.76}{2} = 4.7$ To achieve this value of $V_{th}$ down from $V_{DD} = 9\ V$, we need to adjust the voltage divider configuration to achieve a potentiometer value $k$ = 0.52, as explained in equation 3 previously.

The LM358P IC [4] was chosen to implement the comparator's op-amp, this IC was particularly selected as it features two op-amps on the same chip, meaning more convenient connections and less cost; one of the op-amps was employed to implement the threshold comparator, the other is going to be employed to build a *signal buffer* later in section 3.

It is worth noting that the output voltage of the comparator was measured at $\sim$ 7.5 V (instead of 9 V), one possible reason for this is manufacturing imperfections of the LM358P IC as it introduces internal voltage drops.

Finally, a value for the lamp-post resistor $R_4$ of 330 Ω ± 5% was chosen (instead of 344 Ω) for convenience.

## Section Conclusion

The final schematic for the complete circuit with relevant component values and part numbers is depicted in Figure 11 below.

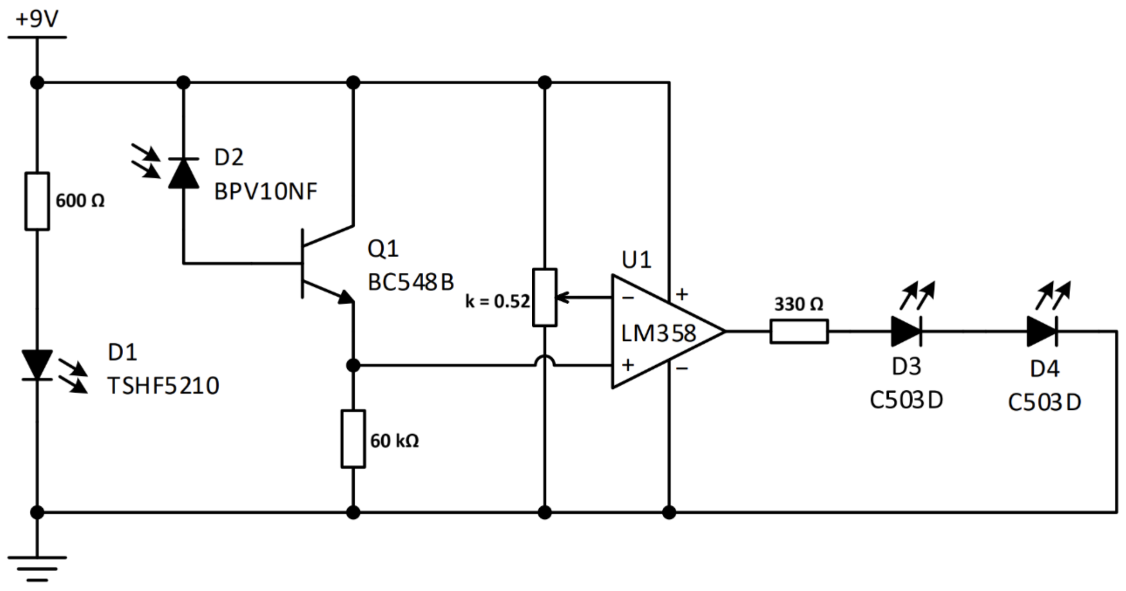


*Figure 11: Completed automatic lamp-post circuit*

The setup effectively activates both C503D LEDs (D3 & D4) with a relative brightness of 0.5 whenever an object is detected within a 10 cm range of the BPV10NF photodiode (D2). Adjusting the 330 Ω resistor allows for varying the brightness of the LEDs, while modifying the potentiometer configuration $k$ alters the sensor's detection range.

# Microcontroller and Output Display

## 15/03/2024

## Introduction & Aims

In this section, a dual-digit, seven-segment output display, as seen in Figure 12, will be connected to a Raspberry Pi Pico microcontroller board. The display will then be programmed to display any number between 0 and 99. This comprises:

- Connect the output display circuit with microcontroller unit (MCU).
- Assigning and configuring the pins on the microcontroller to send signals to the display.
- Creating a function that receives an integer between 0 and 99 as input and outputs signals on the pins so that the display shows that same integer.
- Creating a software that shows the digits 0 through 99 as they are counted and displayed on the output display.

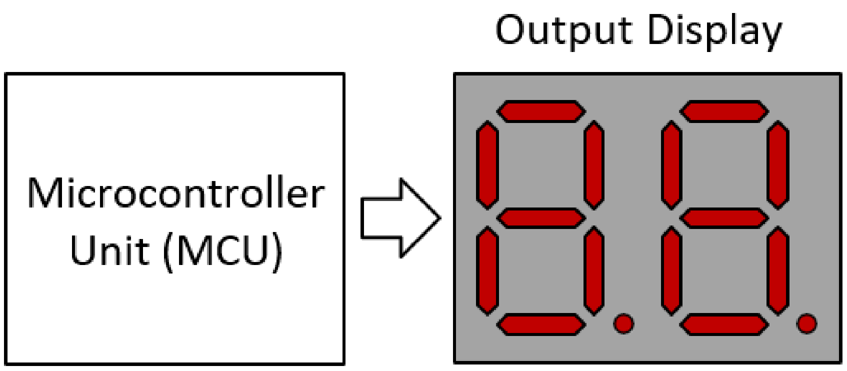


*Figure 12: Raspberry Pi Pico based counter and output display overview.*

## Design Procedure

The program will be designed to perform the following steps in order:

1. Microcontroller and Display Setup:
   - **Objective**: Attach a Raspberry Pi Pico microcontroller to a dual-digit seven-segment display, then set up the system to show digits 0 through 99.
   - **Procedure**:
     a. Hardware Connection: Connect the segments of the Raspberry Pi Pico's dual-digit, seven-segment display to the appropriate GPIO pins using 330Ω resistor for each pin. Ensure that every segment is linked to the appropriate pin as shown in figure 13.

b. Pin Configuration: Set the initial output pin for each pin that is used in the "sq" array. These pins should have their initial state set to low (off).

+5V

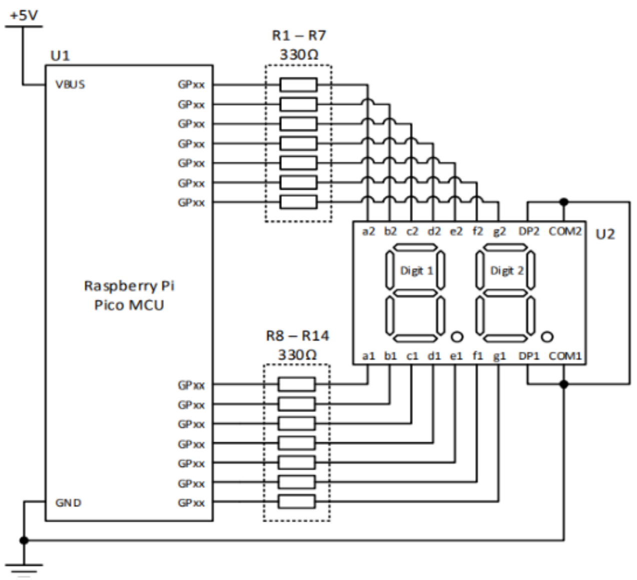


*Figure 13: Microcontroller and output display circuit*

2. Initialization and Setup of Software:
   - **Objective**: Set up the software framework, including function definitions and required variable definitions, to operate the seven-segment display.
   - **Procedure**:

     a. Identify Pin Mappings: Create a two-dimensional list called “sq” (figure 15), in which each sub-list corresponds to a pin arrangement for a single display digit.

     b. State Segment Definitions: Make an array called “segment_states” (figure 16) and put the on/off patterns for all digits 0 through 9 in it.

3. Function Implementation:
   - **Objective**: Create functions that use the input number to manipulate the display.
   - **Procedure**:

     a. “SetDigitSegments()” function (figure 17): The aim is to designate specific digit segments that display a given number while the process is using the value parameter as a guide, decide which segments to light. To display the proper number on the chosen digit (ones or tens), update the GPIO pins as shown in the “segment_states” array.

     b. “SetDisplayDigits()” function (figure 18): The aim of this is to control the entire display by utilizing the given numerical value. Furthermore, the Process is dividing the provided value into ones and tens. For every digit, call “setDigitSegments” function with the appropriate segment values.

4. Loop Built Continuous Display Update:
   - **Objective**: Update the display continuously to cycle through numbers 0 to 99.
   - **Procedure**: “Timer_function” (sleep_interval) function: the target of this funciotn is to manage the periodic modifications to the display by increase the counter from 0 to 99 while updating the display for each number, use a loop. Reset the counter to zero once you've reached 99.

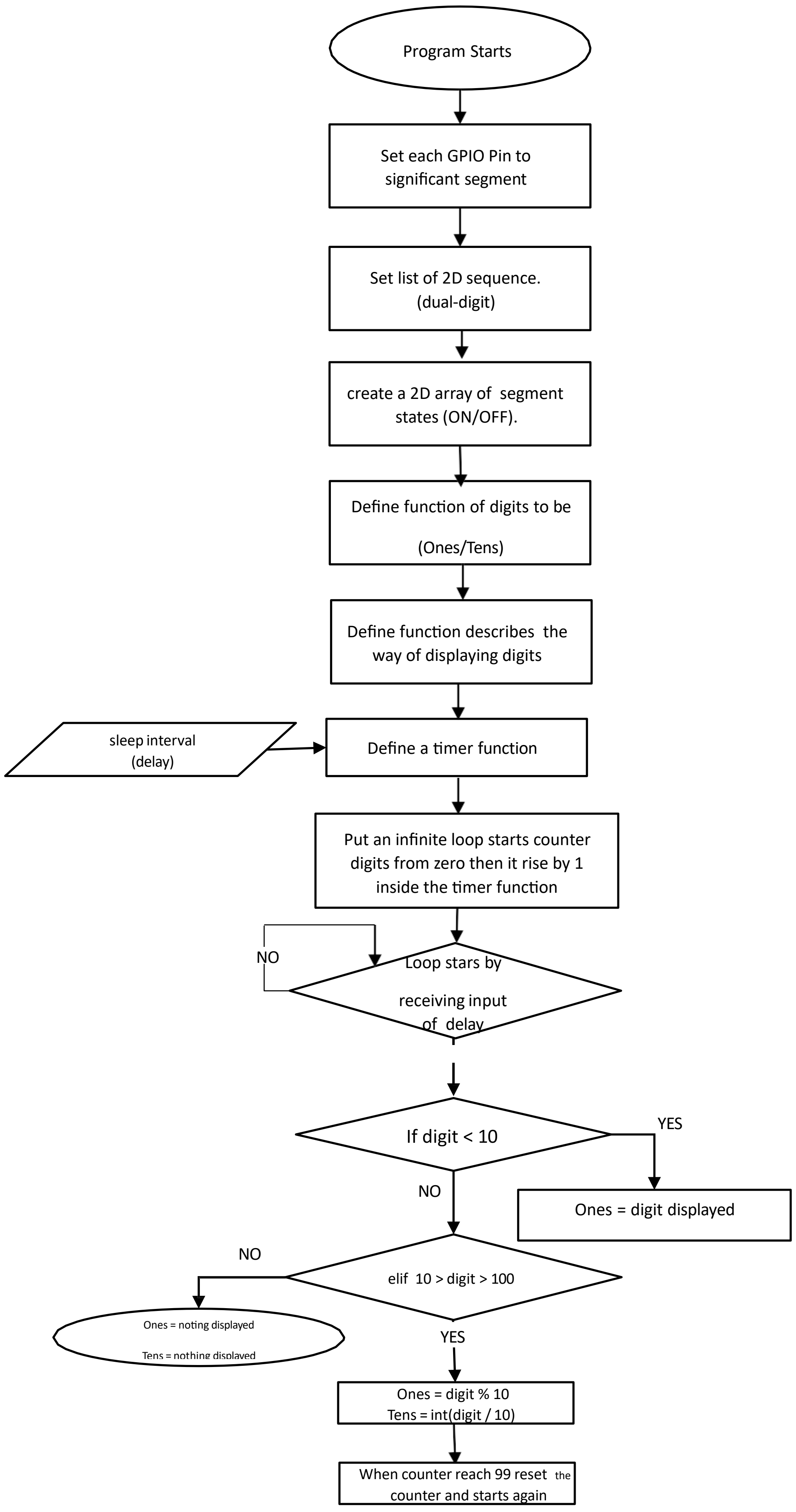

Program Starts
Set each GPIO Pin to significant segment
Set list of 2D sequence. (dual-digit)
create a 2D array of segment states (ON/OFF).
Define function of digits to be (Ones/Tens)
Define function describes the way of displaying digits
sleep interval (delay)
Define a timer function
Put an infinite loop starts counter digits from zero then it rise by 1 inside the timer function
NO
Loop stars by receiving input of delay
If digit < 10
YES
Ones = digit displayed
NO
NO
elif 10 > digit > 100
Ones = noting displayed
Tens = nothing displayed
YES
Ones = digit % 10
Tens = int(digit / 10)
When counter reach 99 reset the counter and starts again

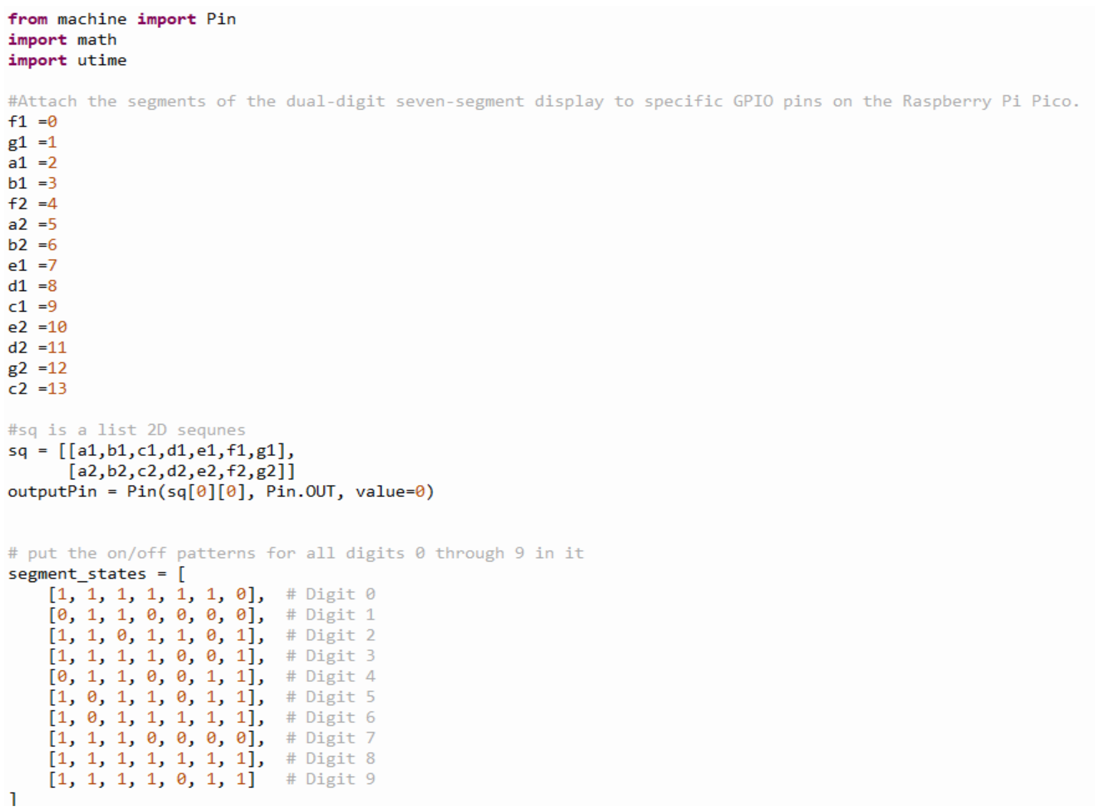
*Figure 14: flowchart showing the operation of the whole program.*

This design procedure outlines the steps to implement and test a program that controls a dual digit seven-segment display connected to a Raspberry Pi Pico microcontroller, ensuring comprehensive coverage from hardware setup to software execution and testing.

## Experimental Procedure and Results

1. Connect the microcontroller to a PC via a micro-USB cable
2. Open the output display program in the Thonny integrated development environment (IDE).
3. test the programme by running the code with different sleep interval to verify that all segments light up appropriately by testing the display for each number between 0 and 99. Also, confirm that the update interval operates as predicted and that the display resets upon reaching 99.
4. Run the program and enter the following intervals in sec to verify that the application handles all cases and displays numbers accurately:
   a. 0.3
   b. 1
   c. 0
   d. 0.01

Below are the code snippets for setup(), loop() with timer_function(), SetDigitSegments() and SetDisplayDigits() that were used in the final program.



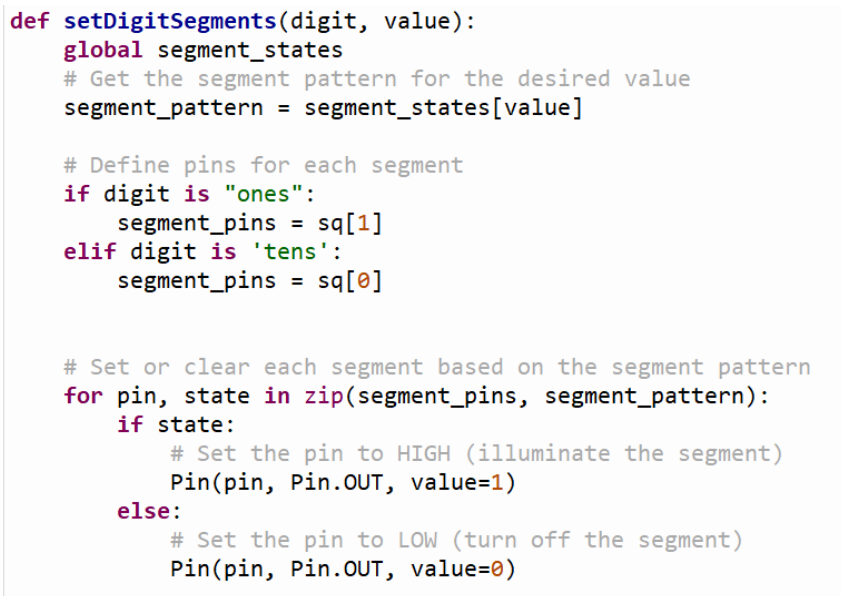
```
from machine import Pin
import math
import utime

#Attach the segments of the dual-digit seven-segment display to specific GPIO pins on the Raspberry Pi Pico.
f1 =0
g1 =1
a1 =2
b1 =3
f2 =4
a2 =5
b2 =6
e1 =7
d1 =8
c1 =9
e2 =10
d2 =11
g2 =12
c2 =13

#sq is a list 2D sequnes
sq = [[a1,b1,c1,d1,e1,f1,g1],
      [a2,b2,c2,d2,e2,f2,g2]]
outputPin = Pin(sq[0][0], Pin.OUT, value=0)


# put the on/off patterns for all digits 0 through 9 in it
segment_states = [
    [1, 1, 1, 1, 1, 1, 0],  # Digit 0
    [0, 1, 1, 0, 0, 0, 0],  # Digit 1
    [1, 1, 0, 1, 1, 0, 1],  # Digit 2
    [1, 1, 1, 1, 0, 0, 1],  # Digit 3
    [0, 1, 1, 0, 0, 1, 1],  # Digit 4
    [1, 0, 1, 1, 0, 1, 1],  # Digit 5
    [1, 0, 1, 1, 1, 1, 1],  # Digit 6
    [1, 1, 1, 0, 0, 0, 0],  # Digit 7
    [1, 1, 1, 1, 1, 1, 1],  # Digit 8
    [1, 1, 1, 1, 0, 1, 1]   # Digit 9
]
```

*Figure 15: setup() code snippet*

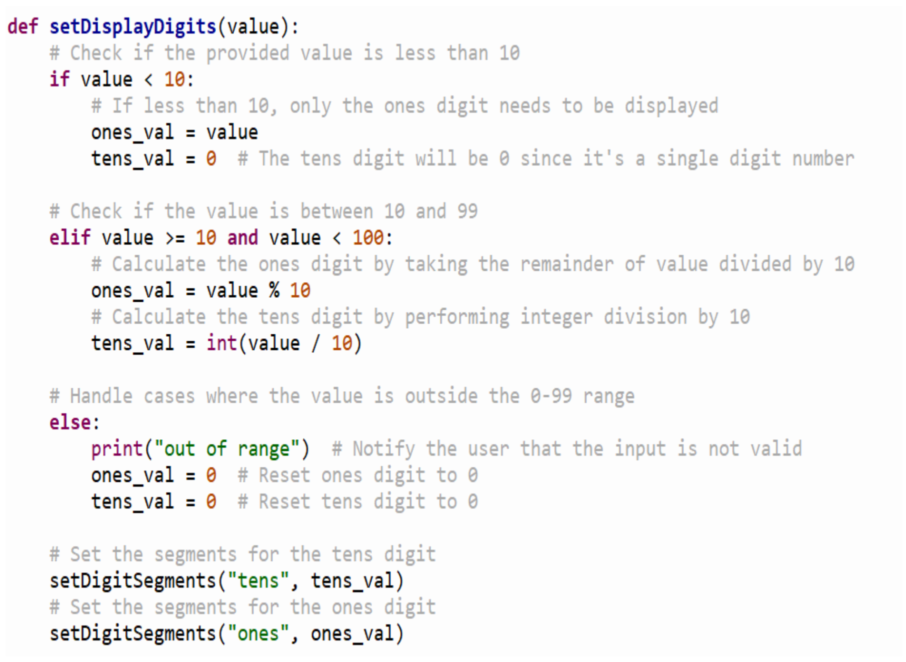


```
def setDigitSegments(digit, value):
    global segment_states
    # Get the segment pattern for the desired value
    segment_pattern = segment_states[value]

    # Define pins for each segment
    if digit is "ones":
        segment_pins = sq[1]
    elif digit is 'tens':
        segment_pins = sq[0]


    # Set or clear each segment based on the segment pattern
    for pin, state in zip(segment_pins, segment_pattern):
        if state:
            # Set the pin to HIGH (illuminate the segment)
            Pin(pin, Pin.OUT, value=1)
        else:
            # Set the pin to LOW (turn off the segment)
            Pin(pin, Pin.OUT, value=0)
```

*Figure 16: SetDigitSegments() code snippet*



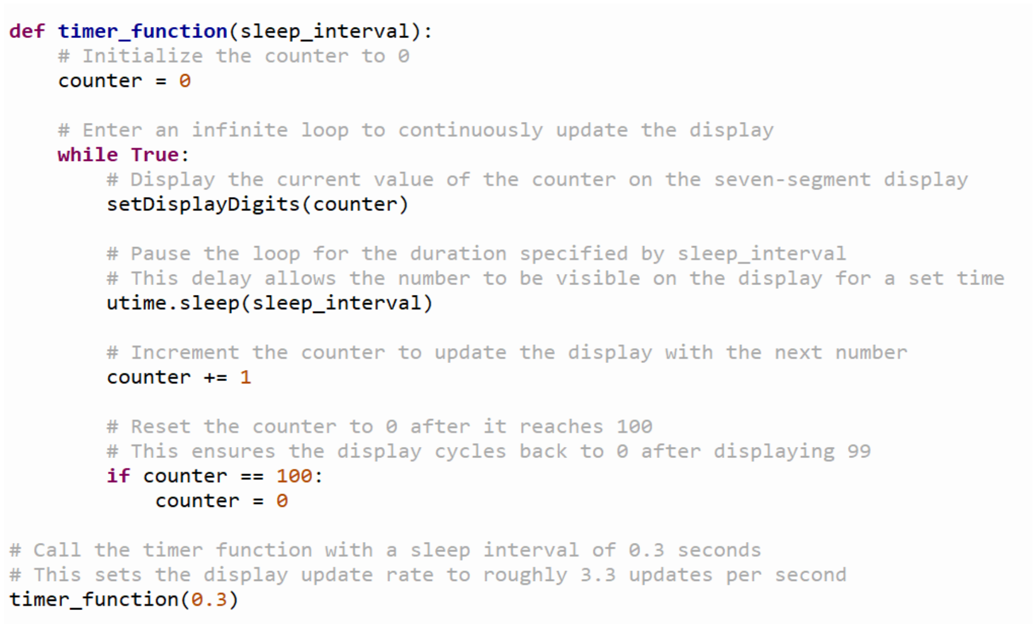
```python
def setDisplayDigits(value):
    # Check if the provided value is less than 10
    if value < 10:
        # If less than 10, only the ones digit needs to be displayed
        ones_val = value
        tens_val = 0  # The tens digit will be 0 since it's a single digit number

    # Check if the value is between 10 and 99
    elif value >= 10 and value < 100:
        # Calculate the ones digit by taking the remainder of value divided by 10
        ones_val = value % 10
        # Calculate the tens digit by performing integer division by 10
        tens_val = int(value / 10)

    # Handle cases where the value is outside the 0-99 range
    else:
        print("out of range")  # Notify the user that the input is not valid
        ones_val = 0  # Reset ones digit to 0
        tens_val = 0  # Reset tens digit to 0

    # Set the segments for the tens digit
    setDigitSegments("tens", tens_val)
    # Set the segments for the ones digit
    setDigitSegments("ones", ones_val)
```

*Figure 17: SetDisplayDigits() code snippet*



```python
def timer_function(sleep_interval):
    # Initialize the counter to 0
    counter = 0

    # Enter an infinite loop to continuously update the display
    while True:
        # Display the current value of the counter on the seven-segment display
        setDisplayDigits(counter)

        # Pause the loop for the duration specified by sleep_interval
        # This delay allows the number to be visible on the display for a set time
        utime.sleep(sleep_interval)

        # Increment the counter to update the display with the next number
        counter += 1

        # Reset the counter to 0 after it reaches 100
        # This ensures the display cycles back to 0 after displaying 99
        if counter == 100:
            counter = 0

# Call the timer function with a sleep interval of 0.3 seconds
# This sets the display update rate to roughly 3.3 updates per second
timer_function(0.3)
```

*Figure 18: loop() in timer_function() code snippet*

Below are the results for each of the tests given of the experimental procedure:

| **Test No.** | **Input** | **Output** | **Success?** |
|---|---|---|---|
| 1 | 0.3 sec sleep interval | Looping through numbers from 0 to 99 with 0.3 sec delay | Y |
| 2 | 1 sec sleep interval | Looping through numbers from 0 to 99 with 1 sec delay | Y |
| 3 | 0 sec sleep interval | All segments lighted up and number 88 displayed | ? |

|  |  |  |  |
|---|---|---|---|
| 4 | 0.01 sec sleep interval | Looping through numbers from 0 to 99 with 0.01 sec delay | Y |

*Table 1: Experimental procedure tests results*

- Tests 1 and 2 and 4 produced the desired outputs.

- Test 3 showed that the program does not work for 0s time interval which could be acceptable However, when the program stops the number 42 is display which means that the running code is working while it is too fast so it is hard to follow each number with a 0s delay, this reasonable as a result of the lower the input value is, the faster looping displayed would appear.

## Section Conclusion

The experimental tests showed:

- The program worked as intended when it loop with 0.01, 1 and 0.3 second sleep interval, however it was not confirmed how many loops that can be handled by the program.
- At a 0 second sleep interval, the application malfunctions and consistently displays the numb er 88, suggesting that abnormally brief intervals could result in display issues.

Recommendations for this work are:

- To improve compatibility with input, rewrite the software to accept and round floating point values before displaying them.

- Improve the program's ability to more thoroughly verify the type and range of inputs. This is k nown as input validation.
  When incorrect inputs are found, send out user alerts to stop such mistakes before they show
  .

- Flexible Ending Conditions: To give the user more control over the display sequence, change the timer_function to enable them to stop the sequence by inputting a certain command, like 'n' and invers the sequence by adding another command.

# Interface Electronics

## Introduction & Aims

## 22/03/2024

This section introduces a higher-level design of the interface that connects the automatic lamp-post circuit and the Raspberry Pi Pico microcontroller unit (MCU) from the previous sections, as shown in Figure 19, this includes:

- Designing a voltage buffer for scaling down the output of the lamp-post driver to a value that can be passed to the MCU.
- Writing an Interrupt Service Routine (ISR) for counting the number of rising edges of the signal received by the lamp-post driver which is then displayed on the two seven-segment displays from section 2.
- Designing a Schmitt trigger to remove unwanted noise occurring when the lamp-post driver output signal changes from low to high (positive edge) and from high to low (negative edge) states.

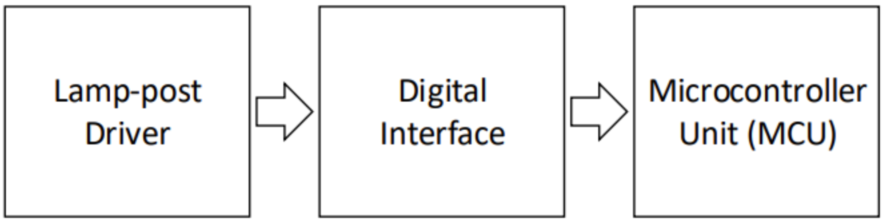


*Figure 19: Higher-level overview of the digital interface.*

The procedure followed here is to start by introducing each additional building block used in this section in the *Theoretical Design* sub-section; the signal buffer, the ISR, and the Schmitt trigger. Then choosing any needed experimental values for the circuit components in the *Experimental Procedure & Results* sub-section. Finally, everything is assembled at the end to create the complete sub-system.

## Theoretical Design

The output value of the lamp-post system from section 1 ranges between 0-0.5 V (when object is out of range) and 7.5-9 V (when object is in range). The RPi MCU, however, only accepts inputs in the range

0-3.3V, thus feeding the output of the sensor directly to the MCU can damage it. One solution for this is to implement a voltage divider, as shown in Figure 20.

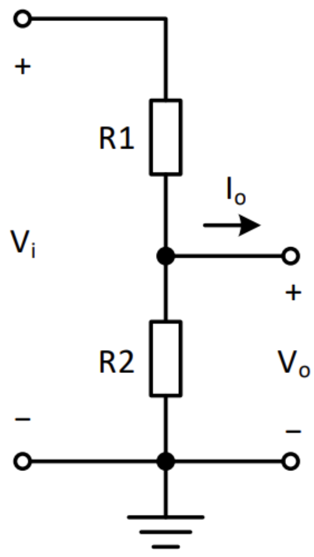


*Figure 20: loaded voltage divider.*

The output voltage of the preceding divider configuration is given by the following equation:

$$V_o = \frac{R_2}{R_1 + R_2} \ (V_i - R_1 I_o) \qquad (5)$$

One issue with this, however, is the dependence of the output voltage $V_o$ on the load current $I_o$, a solution to this problem is to implement a voltage follower consisting of an op-amp operating as a unity-gain amplifier, as shown in Figure 21.

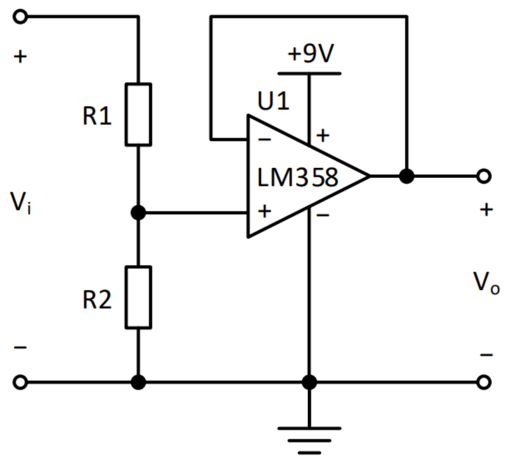


*Figure 21: unity-gain voltage follower.*

The current drawn by the non-inverting terminal of the op-amp is virtually zero ( $I_o \approx 0$ ), therefore:

$$V_+ = \frac{R_2}{R_1 + R_2} \ V_i \qquad (6)$$

The output of the op-amp is fed back to its inverting input which configures it as an amplifier with a gain of one (hence the name *unity-gain*). The result is that the output of the voltage divider is constant and unaltered regardless of the current drawn by the MCU's internal pull-up/pull-down resistor.

The next part is to implement an ISR sub-system described in the flowchart below (Figure 22). Instead of using a timer to appear on the dual-digit display, a program was used to track the low-to-high signals coming from the emitter to the receiver as shown in figure 23 so that they appear on the display, furthermore, the counter resets to 0 whenever the rising-edge count exceeds 10.

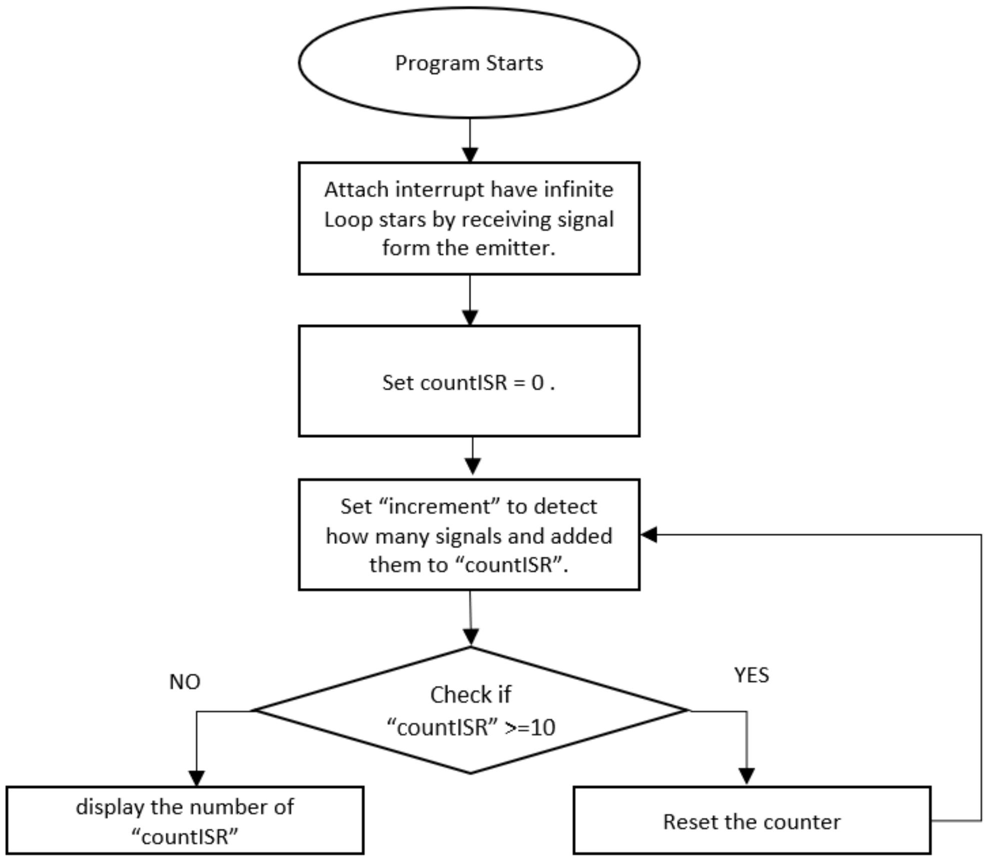
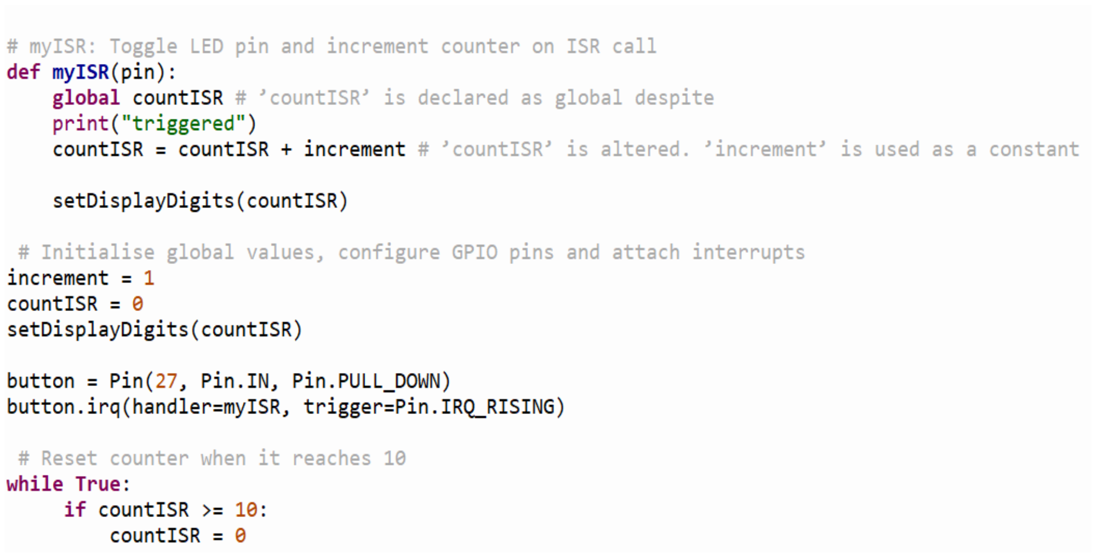

Program Starts
Attach interrupt have infinite Loop stars by receiving signal form the emitter.
Set countISR = 0 .
Set "increment" to detect how many signals and added them to "countISR".
Check if "countISR" >=10
NO
YES
display the number of "countISR"
Reset the counter

*Figure 22: a flowchart for ISR section*



```
# myISR: Toggle LED pin and increment counter on ISR call
def myISR(pin):
    global countISR # 'countISR' is declared as global despite
    print("triggered")
    countISR = countISR + increment # 'countISR' is altered. 'increment' is used as a constant

    setDisplayDigits(countISR)

 # Initialise global values, configure GPIO pins and attach interrupts
increment = 1
countISR = 0
setDisplayDigits(countISR)

button = Pin(27, Pin.IN, Pin.PULL_DOWN)
button.irq(handler=myISR, trigger=Pin.IRQ_RISING)

 # Reset counter when it reaches 10
while True:
     if countISR >= 10:
         countISR = 0
```

*Figure 23: ISR program used to detect and count the signals.*

The final part is to implement a Schmitt trigger circuit to resolve electrical noise issues affecting the threshold comparator discussed in section 1. The comparator's oscillation between high and low states as shown in Figure 24, caused by the noise, leads the microcontroller to count unintended rising edges, thus inaccuracy in the displayed count.

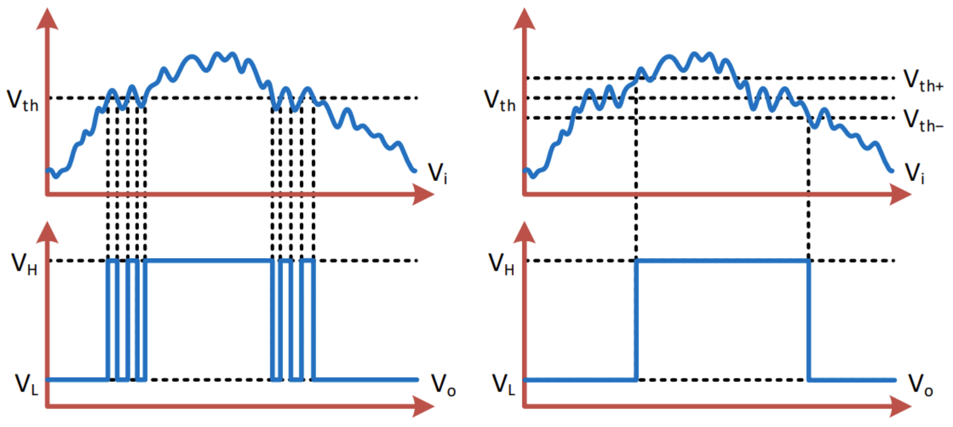
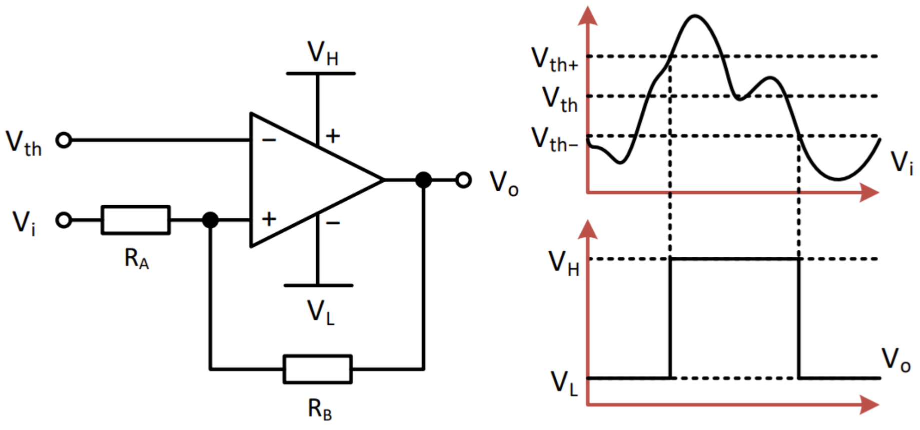


*Figure 24: Effect of electrical noise on types of comparators; threshold comparator (left), Schmitt trigger (right)*

The Schmitt trigger is effectively a comparator with hysteresis - the threshold voltage will change depending on the state of the output (Figure 25b), that is, the output toggles when the input is $V_i > V_{th+}$ and previous state is $V_o = V_L$, or if the input is $V_i < V_{th-}$ and previous state is $V_o = V_H$.



*(a) Circuit schematic* *(b) operational waveforms*

*Figure 25: Operational-Amplifier Schmitt Trigger operation*

The circuit schematic of the Schmitt trigger is shown in Figure 25a. The value of the resistors $R_A$, and $R_B$, determine the magnitudes of $V_{th+}$ and $V_{th-}$ according to the following equations:

$$V_{th+} = V_{th} + \frac{R_A}{R_B}(V_{th} - V_L) \qquad (7)$$

$$V_{th-} = V_{th} - \frac{R_A}{R_B}(V_H - V_{th}) \qquad (8)$$

The values for $R_A$ and $R_B$ are to be found experimentally in the next subsection. Equations 6 & 7 indicate that small values of the ratio $\frac{R_A}{R_B}$ result in the circuit being more sensitive to electrical noise while large values being more immune.

## Experimental Procedure & Results

Before assembling the preceding building blocks of the complete system, suitable values for resistors $R_1$, $R_2$ (signal buffer), $R_A$, and $R_B$ (Schmitt trigger) need to be determined experimentally. Note that the ratios $\frac{R_1}{R_2}$ and $\frac{R_A}{R_B}$ are what matter rather than the exact values for each of the resistors. Thus, two potentiometers of configurations $k_1$ and $k_2$ (Figure 7) respectively were employed instead of fixed resistors for convenience.

Recall that the output of the lamp-post circuit was measured to be $\sim$ 7.5 V, instead of 9 V ($V_{DD}$). Thus, the value $k_1$ need to be chosen such to achieve an output voltage of 3.3 V (the range of the MCU) when the divider is fed an input of 7.5 V, plugging these values in equation 3 yields:

$$k_1 = \frac{3.3}{7.5} = 0.44 \,.$$

The potentiometer used for the Schmitt trigger circuit was of a value of R = 100 kΩ. The value $k_2$ is to be chosen such to allow the system to count each individual occurrence of an object coming into the range of the proximity sensor. By trial and error, a suitable value was chosen to be $k_2$ = 0.17. Recall that $R_A = k_2 R$, and $R_B = (1 - k_2)R$ (Figure 7), this yields:

$$\frac{R_A}{R_B} = \frac{k_2}{1 - k_2} = \frac{0.17}{1 - 0.17} = 0.21$$

Plugging this value, along with $V_{th}$ = 4.7, $V_H = 9$ and $V_L$ = 0, in equations 7, and 8, yields: $V_{th+} = 5.66\,V$ , $V_{th-} = 3.81\,V$.

Note that the threshold comparator from section 1 is replaced by the Schmitt trigger circuit, both signal buffer and the Schmitt trigger were implemented on the same LM358P IC from previously, as it features two op-amps fabricated on the same chip which comes at a lower cost than two separate single-op-amp ICs.

## Section Conclusion

The final schematic for the complete circuit of the proximity-sensor system with relevant component values and part numbers is depicted in Figure 26 below.

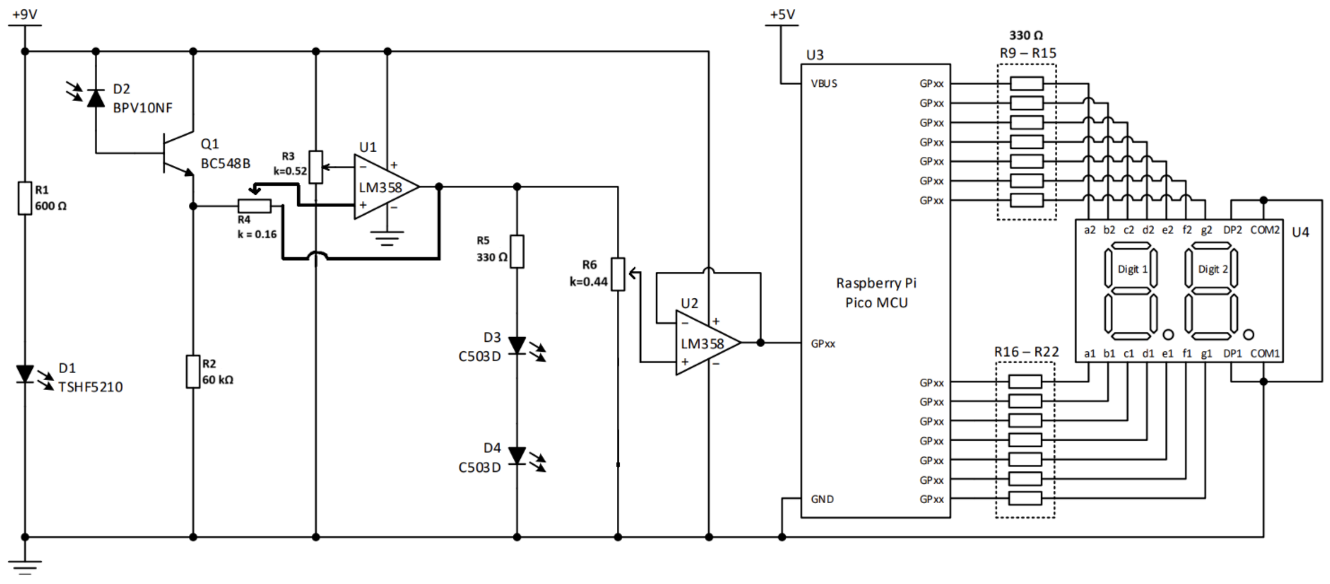


*Figure 26: Completed Automatic lamp-post with counter circuit*

The setup successfully activates both C503D LEDs (D3 & D4) with a relative brightness of 0.5 whenever an object is detected within a 10 cm range of the BPV10NF photodiode (D2). Moreover, it counts the number of times an object is present in the range and displays the count on a dual-digit, 7-segment display. Furthermore, The counter zero-resets whenever it exceeds 10. Adjusting resistor $R_5$ allows for varying the brightness of the LEDs, while modifying potentiometer $R_3$ configuration alters the sensor's detection range. Finally, the system immunity to electrical noise is determined by potentiometer $R_4$ with a proportionality to its $k$ value.

One issue with this design, however, is the extra non-essential power consumption of the signal buffer system (U2 & R6). One enhanced design could be by replacing it with an N-type MOSFET; with its base terminal connected to the output of U1, and the source-drain channel to be formed between one of the RPi Pico 3V3 output pins, and the already-employed GPI pin (GPIO 27), as depicted in Figure 27.

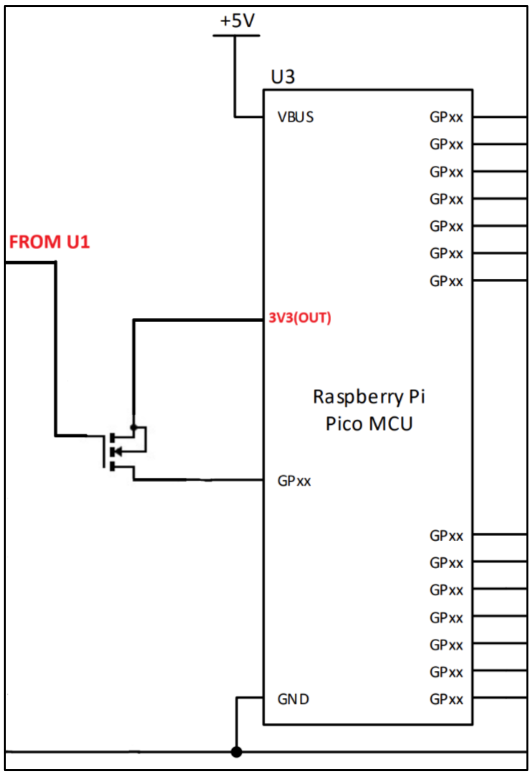


*Figure 27: Enhanced design of the sensor digital interface by adding an N-MOS*

Note that the chosen N-MOS should be of a specification that ensures the output of U1 (7.5–9 V ON) to be above the base threshold voltage and is within its functional range. One recommendation is Infineon's IRFZ34NPbF (datasheet 7)

# Datasheets

[1] “TSHF5210: High speed infrared emitting diode, 890 nm, GaAlAs double hetero,” Datasheet, Vishay Semiconductors, Aug. 2011. [Online]. Available: https://www.vishay.com/docs/81313/ tshf5210.pdf

[2] “BPV10NF: Silicon PIN photodiode,” Datasheet, Vishay Semiconductors, Nov. 2011. [Online]. Available: https://www.vishay.com/docs/81503/bpv10nf.pdf

[3] “BC54(6,7,8)(A,B,C): Amplifier transistors NPN silicon,” Datasheet, ON Semiconductor, Jun. 2012. [Online]. Available: https://www.onsemi.com/pub/Collateral/BC546-D.PDF

[4] “LMx58-N: Low-power, dual-operational amplifiers,” Datasheet, Texas Instruments, Dec. 2014. [Online]. Available: https://www.ti.com/lit/ds/symlink/lm358-n.pdf?ts=1607698106707&ref_url=https%253A%252F%252Fwww.ti.com%252Fproduct%252FLM358-N

[5] “C503D-WAN: Cree 5-mm round LED,” Datasheet, Cree Inc., 4600 Silicon Drive, Durham, NC 27703, 2011. [Online]. Available: https://www.cree.com/led-components/media/ documents/C503D-WAN-935.pdf

[6] “HDSP-52x(E,G,Y) series: 14.2 mm (0.56 inch) general purpose two digit seven segment displays,” Datasheet, Avago Technologies, Jun. 2012. [Online]. Available: https://docs.broadcom.com/doc/AV02-3586EN

[7] “IRFZ34NPbF HEXFET ® Power MOSFET V DSS = 55V R DS(on) = 0.040Ω.” Accessed: Apr. 28, 2024. [Online]. Available: https://www.mouser.co.uk/datasheet/2/196/Infineon_IRFZ34N_DataSheet_v01_01_EN-3363377.pdf