



TECHNICAL REPORT

SLOT CAR AUTOMATIC LAMP-POST AND COUNTING SYSTEM

Bexultan Khabiyev and Damjan Janchevski

Department of Electrical and Electronic Engineering, School of Computer Science, Electrical and
Electronic Engineering, and Engineering Maths (SCEEM), University of Bristol

Contents

Lab 1 – Analogue Electronics	2
Aims	2
Design Procedure	2
Experimental Procedure and Results	6
Conclusions and Recommendations	12
Lab 2 – Microcontroller and Output Display.....	13
Aims	13
Design Procedure	13
Experimental Procedure and Results	20
Conclusions and Recommendations	23
Lab 3 – Interface Electronics	24
Aims	24
Design Procedure	24
Experimental Procedure and Results	29
Conclusions and Recommendations	29

Lab 1 – Analogue Electronics

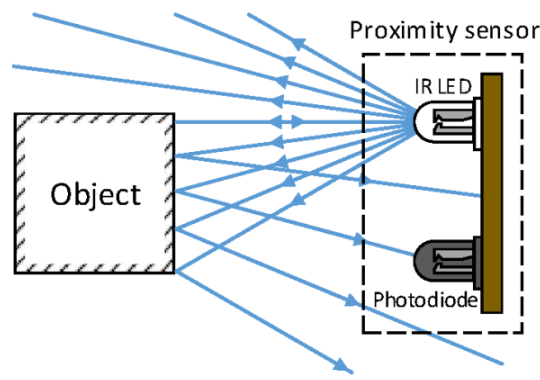
31/01/2023

Aims

- Design and build the experimental transmitter circuit on a breadboard using the following components:
 - Infrared (IR) LED
 - 100Ω resistor
 - $1k\Omega$ potentiometer
 - +9V battery
- Assemble the experimental receiver circuit on the breadboard, next to the transmitter circuit, using the components presented in the list below:
 - Photodiode
 - Bipolar junction transistor (BJT)
 - $100k\Omega$ potentiometer
 - +9V battery
- Characterise and tune both the transmitter and receiver circuits by replacing the resistive elements with a resistor of an appropriate value.
- Build the threshold comparator using an op-amp (LM358) and a $10k\Omega$ potentiometer.
- Assemble the lamp-post light-emitting diodes and select a resistance value to achieve a relative luminous intensity of 0.5 for each LED.

Design Procedure

In this study, the proximity sensor was utilised for the detection of nearby objects that can activate an automatic lamp-post circuit. It is composed of two components, namely, the transmitter and the receiver. The positioning of the IR LED and photodiode is such that when an object is in close proximity, the radiation emitted by the IR LED is reflected back to the photodiode, as depicted in Figure 1.



1

Figure 1. Proximity sensor operation overview

¹ Lab1 – Analog Electronics, Dr Ian Laird, 2023

The transmitter circuit comprises an IR LED in series with a resistor, as Figure 2 depicts. The resistor is responsible for regulating the current flowing through the IR LED, ensuring that it remains within the maximum allowable limit, thus safeguarding it from overheating. In addition, by adjusting the potentiometer we may vary the radiant power level of the IR LED, and therefore control the strength of the emitted IR signal.

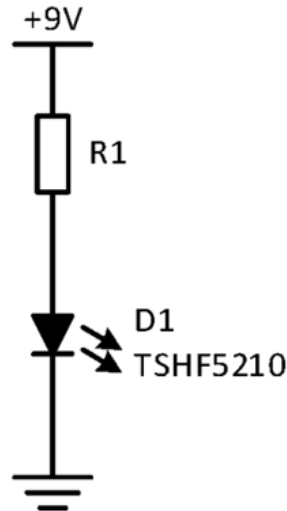


Figure 2. Operational transmitter circuit schematic

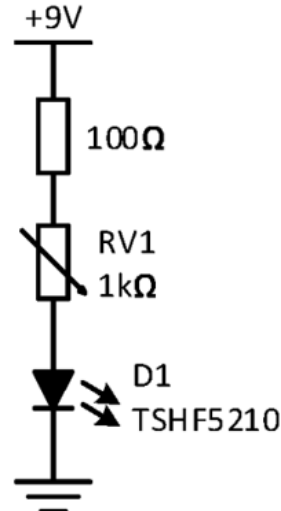


Figure 3. Experimental transmitter circuit schematic

The receiver circuit is composed of a photodiode, a bipolar junction transistor (BJT), and a resistor, connected in the manner illustrated in Figure 4. The BJT and resistor function as a current gain amplifier. When exposed to IR radiation, the photodiode produces a current supplied to the BJT base.

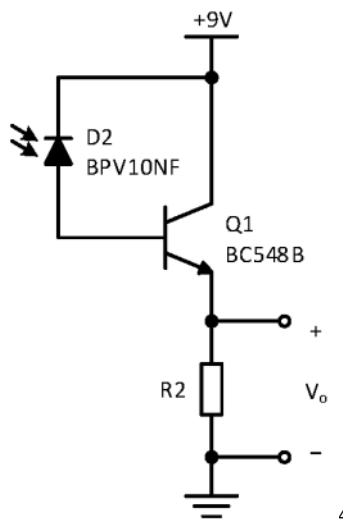


Figure 4. Operational receiver circuit schematic

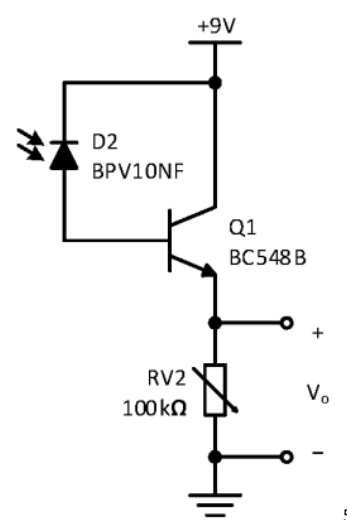


Figure 5. Experimental receiver circuit schematic

The lamp-post driver subsystem employs a threshold comparator circuit, which is made up of an op-amp and potentiometer connected in the configuration demonstrated in Figure 8. The

² Lab1 – Analog Electronics, Dr Ian Laird, 2023

³ Lab1 – Analog Electronics, Dr Ian Laird, 2023

⁴ Lab1 – Analog Electronics, Dr Ian Laird, 2023

⁵ Lab1 – Analog Electronics, Dr Ian Laird, 2023

comparator operates by comparing the non-inverting input V_+ to the inverting input V_- to determine whether the output V_o should be set to the positive power rail (V_H) or the negative power rail (V_L). Figure 7 illustrates an instance of output behaviour during comparator operation, which can be summarized as follows:

$$\text{If } V_+ > V_- \text{ then } V_o = V_H$$

$$\text{If } V_+ < V_- \text{ then } V_o = V_L$$

Thus, if an object comes close enough to the proximity sensor, the circuit produces a voltage that exceeds the threshold voltage established by the potentiometer, triggering the lamp-post light.

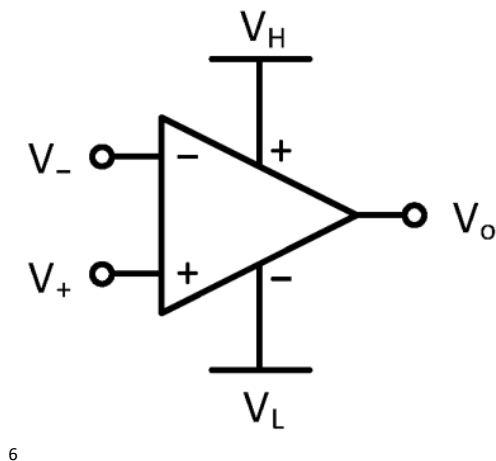


Figure 6. Operational-amplifier comparator circuit schematic

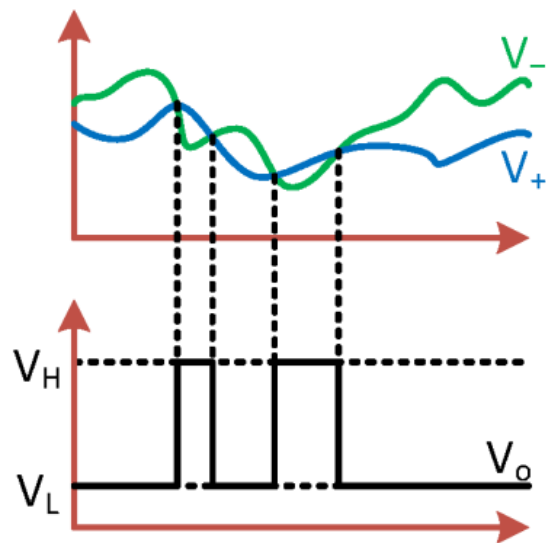
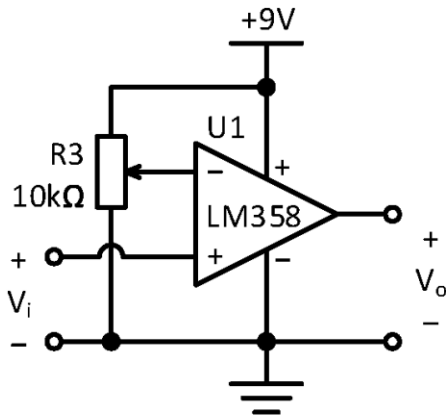


Figure 7. Operational waveforms

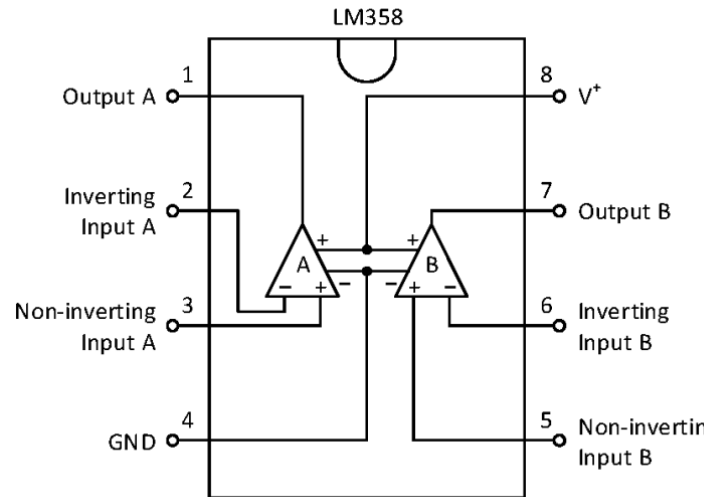
⁶ Lab1 – Analog Electronics, Dr Ian Laird, 2023

⁷ Lab1 – Analog Electronics, Dr Ian Laird, 2023



8

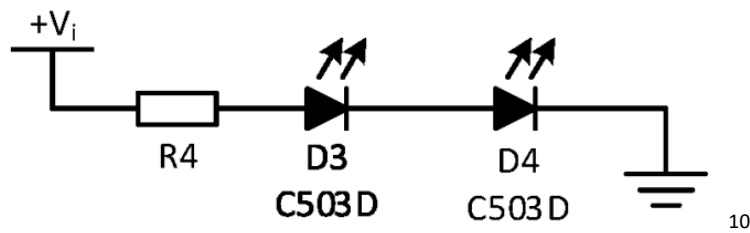
Figure 8. Threshold comparator



9

Figure 9. LM358 op-amp connections

The lamp-post light is comprised of two light-emitting diodes (LEDs) and a resistor, as depicted in Figure 10. Similar to the transmitter circuit, the resistor's function is to regulate the brightness of the LEDs and restrict the current flowing through them to stay below the maximum threshold.



10

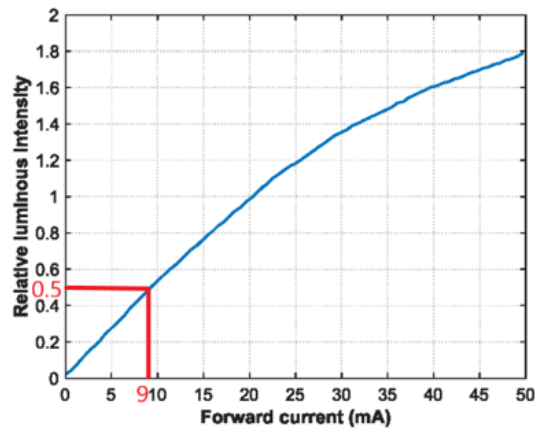
Figure 10. Lamp-post LEDs circuit

By analysing the graphs in Figure 11 and Figure 12, the required resistor value that will give a relative luminous intensity of 0.5 for each LED was calculated.

⁸ Lab1 – Analog Electronics, Dr Ian Laird, 2023

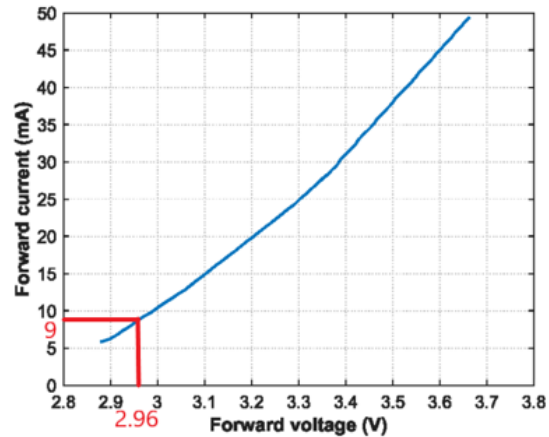
⁹ Lab1 – Analog Electronics, Dr Ian Laird, 2023

¹⁰ Lab1 – Analog Electronics, Dr Ian Laird, 2023



11

Figure 11. Relative luminous intensity vs Forward current



12

Figure 12. Forward current vs Forward voltage

The Ohm's law states that:

$$R = \frac{V}{I} \quad (1)$$

Where V – forward voltage, I – forward current.

Substituting the values of voltage and current derived from Figure 12 into (1) gives:

$$R = \frac{2.96}{9.00 \times 10^{-3}} \approx 329\Omega$$

07/02/2023

Experimental Procedure and Results

To build the transmitter circuit, it was required to do the actions below:

1. Insert the IR LED onto the breadboard, with the anode (positive terminal) on one side and the cathode (negative terminal) on the other side. Ensure that the LED is oriented correctly, with the longer leg representing the anode.
2. Place a 100Ω resistor onto the breadboard.
3. Connect one end of the 100Ω resistor to one of the outer terminals of the potentiometer.
4. Place the middle terminal of the potentiometer in a way so it connects to the anode of the LED.
5. Connect the positive terminal of the +9V battery to the same row as the resistor.
6. Connect the negative terminal of the +9V battery to the same row as the cathode of the LED.

¹¹ Lab1 – Analog Electronics, Dr Ian Laird, 2023

¹² Lab1 – Analog Electronics, Dr Ian Laird, 2023

Upon completing the transmitter circuit, we conducted measurements on the forward voltage and current of the IR LED at ten different resistance values of the potentiometer. These measurements were used to yield the following graphs:

- Diode forward voltage as a function of the potentiometer resistance
- Diode forward current as a function of the potentiometer resistance
- Diode forward current as a function of the diode forward voltage

To measure the component values, the following steps were taken:

1. One multimeter was connected in series with the potentiometer and set to measure resistance.
2. The resistance of the potentiometer was reduced to its lowest point by adjusting the position of the sliding contact.
3. As soon as the potentiometer's resistance value was recorded, a +9V battery's power was supplied to the circuit for measuring voltage and current.
4. The ends of another multimeter were placed in parallel with the diode to record the voltage across it.
5. The multimeter was disconnected and its ends were placed in series with the diode to register the current flowing toward it.
6. The resistance of the potentiometer was increased and the new resistance value is recorded.
7. The +9V battery from the circuit is disconnected from the circuit.
8. Steps 3-7 were repeated for variable resistor values between 0-1000 Ω while ensuring a 50-150 Ω difference between each recorded resistance value.

The required component values are recorded in Table 1 and displayed as graphs in Figure 13, Figure 14, and Figure 15.

Table 1. Experimental measurements for the transmitter's circuit

Resistance (Ω)	Voltage (V)	Current (mA)
0.9	1.57	60.84
144	1.33	28.37
214	1.322	22.73
327	1.309	16.89
485	1.306	12.83
588	1.296	11.04
697	1.289	9.6
796	1.283	8.6
899	1.28	7.76
983	1.278	7.19

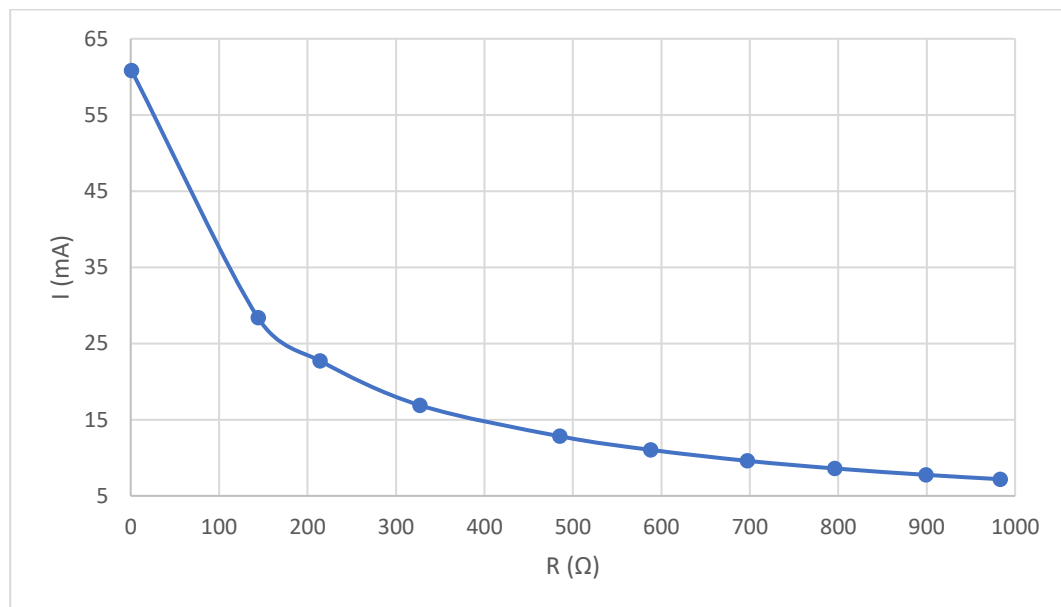


Figure 13. Diode forward voltage as a function of the potentiometer resistance

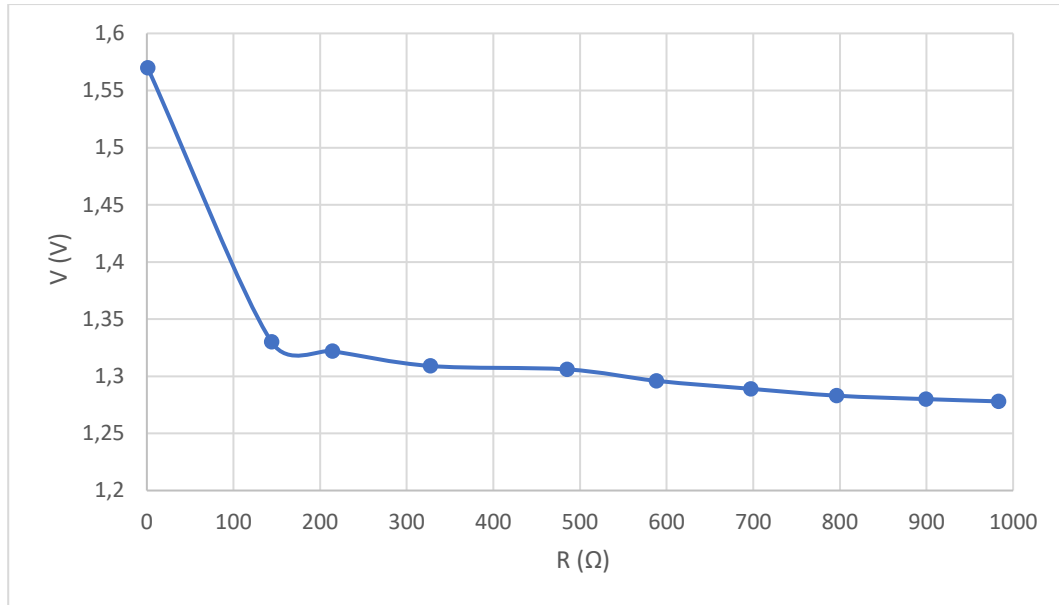


Figure 14. Diode forward current as a function of the potentiometer resistance

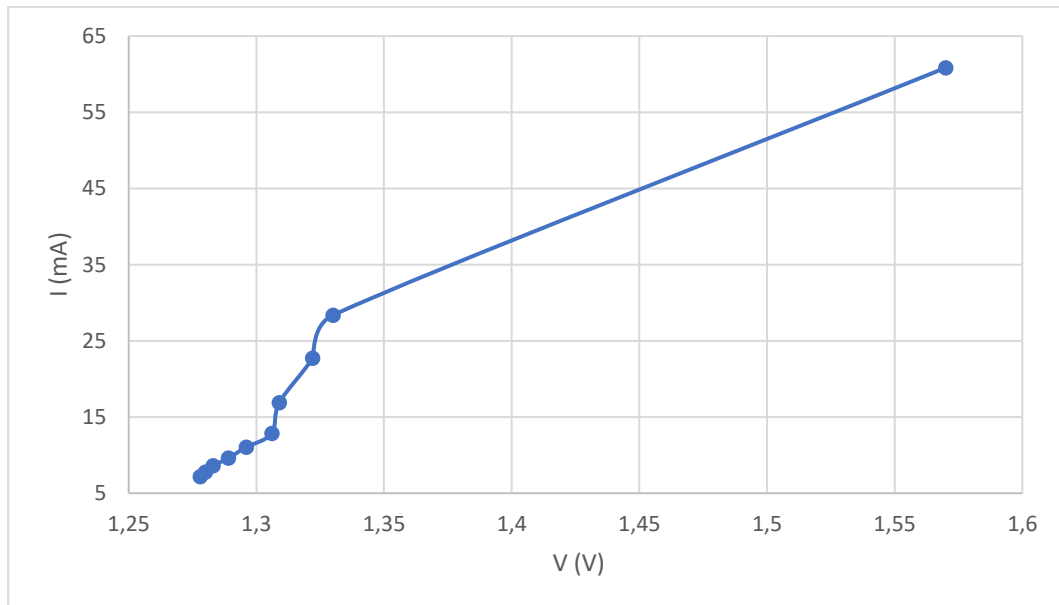


Figure 15. Diode forward current as a function of the diode forward voltage

To assemble the receiver circuit on a breadboard, the following steps were taken:

1. Place the photodiode on the breadboard and connect its cathode (shorter lead) to the positive terminal of the +9V battery and its anode (longer lead) to the base of the BJT.
2. Connect the collector of the BJT to the positive power supply (+Vcc) and the emitter to the negative power supply (GND).
3. Connect the potentiometer between the base of the BJT and ground. The potentiometer can be used to adjust the bias voltage at the base of the BJT.

Next, the complete proximity sensor (transmitter and receiver) was characterised by doing the following:

1. Measure and record the resistance of the potentiometers in both the transmitter and receiver circuits.

2. Ensure that there are no objects in proximity to the sensor, power up the circuit, and measure the output voltage of the receiver circuit.
3. With the circuit still powered, place a hand at a fixed distance (between 5 to 10 cm) from the proximity sensor and measure the output voltage again.
4. Remove the power from the circuit.
5. Repeat steps 1 to 4 for the following resistance values: 0Ω , 251Ω , 503Ω , and 983Ω in the transmitter circuit and $25k\Omega$, $50k\Omega$, $75k\Omega$, and $100k\Omega$ in the receiver circuit. Cover all 20 possible combinations based on the chosen resistance values.
6. For each measurement, calculate the difference between the voltage measured when an object is within range of the sensor and when there is none (i.e., the difference between the measurements obtained in steps 2 and 3).

The recorded component values are given in Table 2, along with the calculated difference between the output voltage when there is an object close to the proximity sensor and the output voltage when there is no object close to the sensor. The voltage change as a function of the receiver circuit's potentiometer resistance was plotted using this data, as shown in Figure 16.

Table 2. Experimental values for the proximity sensor's characterisation

Resistance, transmitter (Ω)	Resistance, receiver (Ω)	Voltage, without an object (V)	Voltage, with an object (V)	Voltage change (V)
0	25	1.09	8.49	7.4
0	50	2.08	8.55	6.47
0	75	3.09	8.59	5.5
0	100	4.42	8.61	4.19
251	25	0.5	3.37	2.87
251	50	0.96	5.97	5.01
251	75	1.45	8.1	6.65
251	100	2.01	8.64	6.63
503	25	0.52	2.53	2.01
503	50	1.03	5.34	4.31
503	75	1.56	6.75	5.19
503	100	2.13	8.65	6.52
750	25	1.15	2.95	1.8
750	50	2.3	9.18	6.88
750	75	3.35	8.66	5.31
750	100	4.6	8.72	4.12
983	25	0.56	1.55	0.99
983	50	1.08	3.63	2.55
983	75	1.63	4.43	2.8
983	100	2.25	5.49	3.24

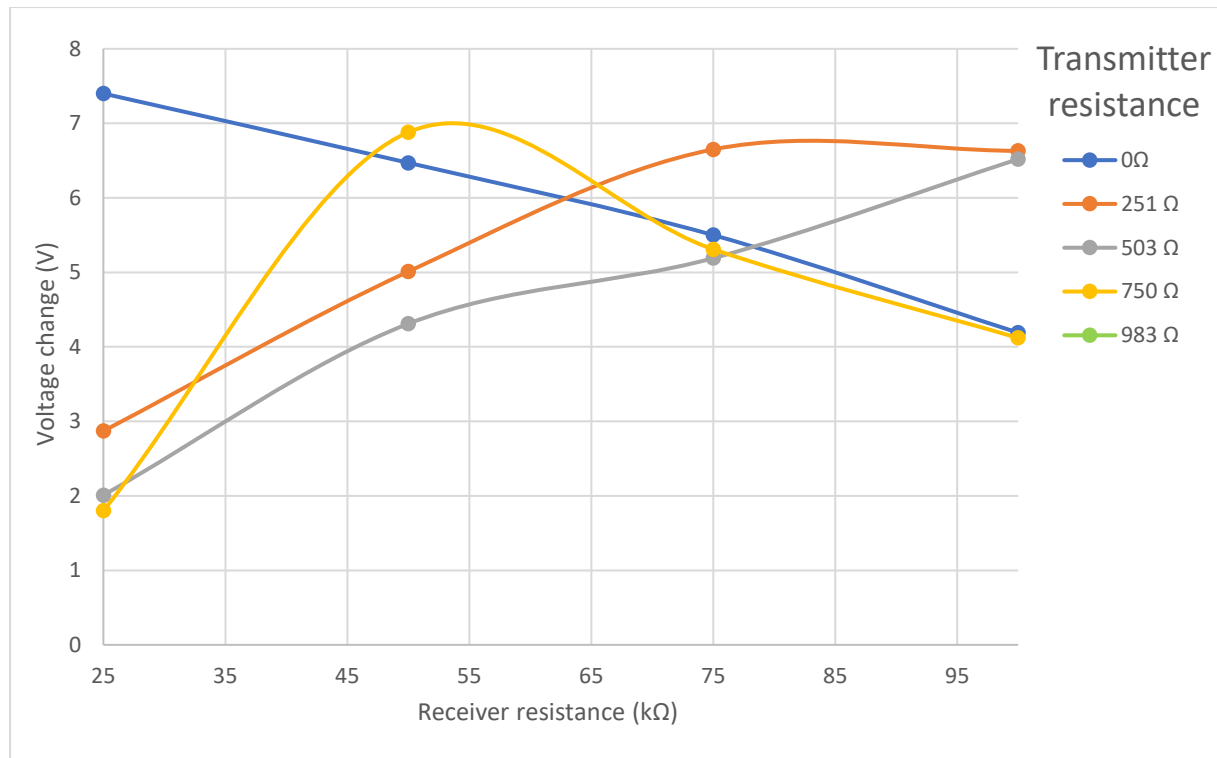


Figure 16. Voltage change as a function of the receiver circuit's potentiometer resistance

The results of the experiment indicate that the highest voltage change was observed when the potentiometer in the transmitter had almost zero-resistance value, leaving only a 100Ω resistor in the circuit, and the potentiometer in the receiver was set to 25kΩ, as evidenced by the data plotted on Figure 16.

After that, the threshold comparator was assembled using an op-amp (LM358) and a 10kΩ potentiometer in the following way:

1. Place the LM358 op-amp on the breadboard and connect its V+ pin to the positive power supply (+Vcc) and its ground (GND) to the negative power supply.
2. Connect the non-inverting input pin of the LM358 op-amp to the output of the proximity sensor circuit.
3. Connect the potentiometer between the inverting input pin of the LM358 op-amp and ground. The potentiometer can be used to adjust the threshold voltage of the circuit.

Conclusions and Recommendations

Overall, the laboratory team has successfully designed and assembled the analogue electronics that comprise the automatic lamp-post circuit.

In particular, the proximity sensor - comprising a transmitter and a receiver - has undergone thorough characterization and calibration to detect nearby objects that could trigger the automatic lamp-post circuit. Through a series of experiments, the resistance values of the potentiometers in the proximity sensor were alternated while recording the corresponding voltage change when an object was exposed to the transmitter circuit's infrared radiation and when it was not. The resulting data was used to generate a graph that allowed for analysis of

the sensor's performance. Based on the data, it is recommended that the transmitter circuit use a sole resistive element of 100Ω , while the potentiometer in the receiver circuit should be replaced with a $25k\Omega$ resistor to optimize the response of the proximity sensor.

During the assembly of the lamp-post light, a 330Ω resistor was incorporated in series with two Light Emitting Diodes (LEDs) to attain a relative luminous intensity of 0.5 for each LED. Although the theoretical calculations for this section of the experiment yielded a resistance of 329Ω , a decision was made to employ a single 330Ω resistor to maintain simplicity in the circuit.

To further enhance the accuracy and reliability of future experiments, it is recommended to perform measurements in a controlled environment with minimal external interference. While measuring the voltage change in the proximity sensor, the circuit was subjected to a substantial amount of artificial light emanating from the ceiling lamps, including an infrared radiation component in its spectrum. The presence of this radiation may have had a potential impact on the precision of the experiment. Therefore, it is suggested to perform measurements in an environment with zero external infrared radiation to eliminate any potential interference and ensure the accuracy of the results.

Lab 2 – Microcontroller and Output Display

14/02/2023

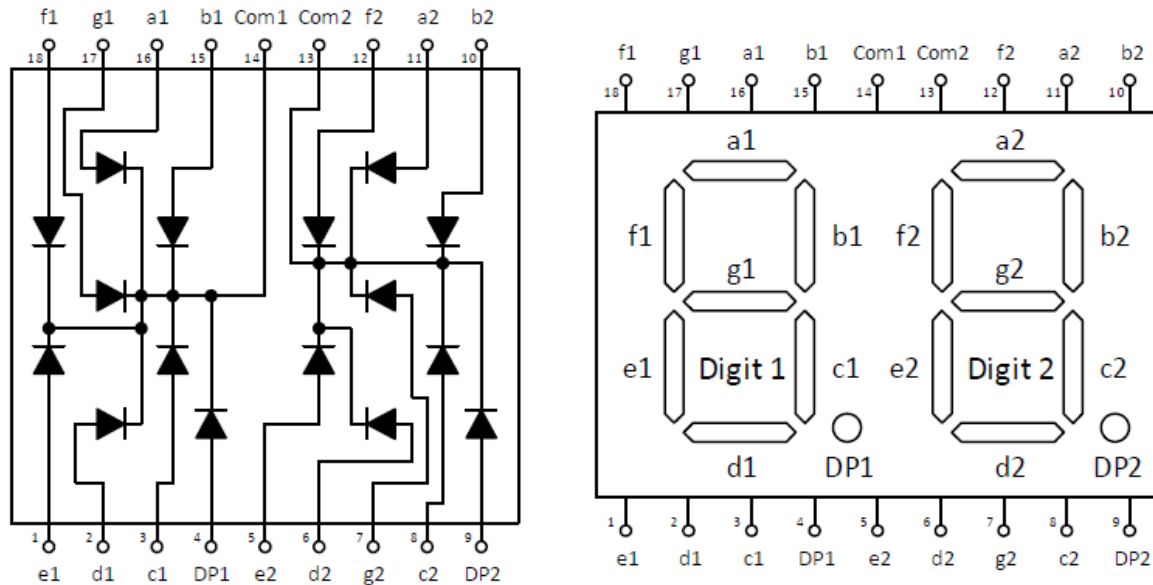
Aims

- Design and build the output display hardware based on the HDSP-523E dual-digit 7-segment display and controlled by a Raspberry Pi Pico MCU.
- Create a program that:
 - Controls the output of the two 7-segment digit display
 - Outputs a number between 0 and 99
 - Counts from 0 to 99

Design Procedure

The output display is an HDSP-523E dual-digit, 7-segment light-emitting diode (LED) display. The HDSP-523E is an 18-pin device that consists of 16 LEDs arranged as shown in Figure 17. The LEDs are divided into the following groups:

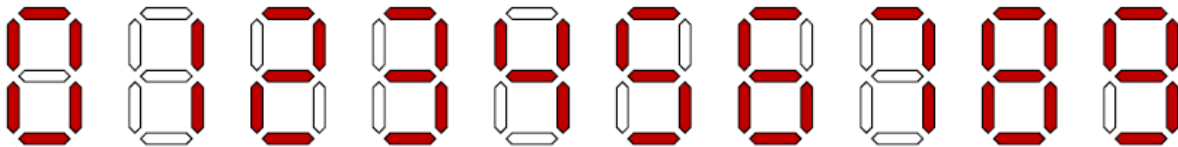
- 7 LEDs are used to display the "tens" digit (digit 1, a1 to g1)
- 7 LEDs are used to display the "ones" digit (digit 2, a2 to g2)
- 1 LED is used to display the decimal point after digit 1 (DP1)
- 1 LED is used to display the decimal point after digit 2 (DP2)



13

Figure 17. HDSP-523E display connector pinout

Figure 18 shows how all digits from 0 to 9 can be displayed by illuminating certain combinations of the 7 LEDs designated for one of the digits. These combinations are called 7-segment digit states. Hence, the HDSP-523E can be used to display any two-digit number.



14

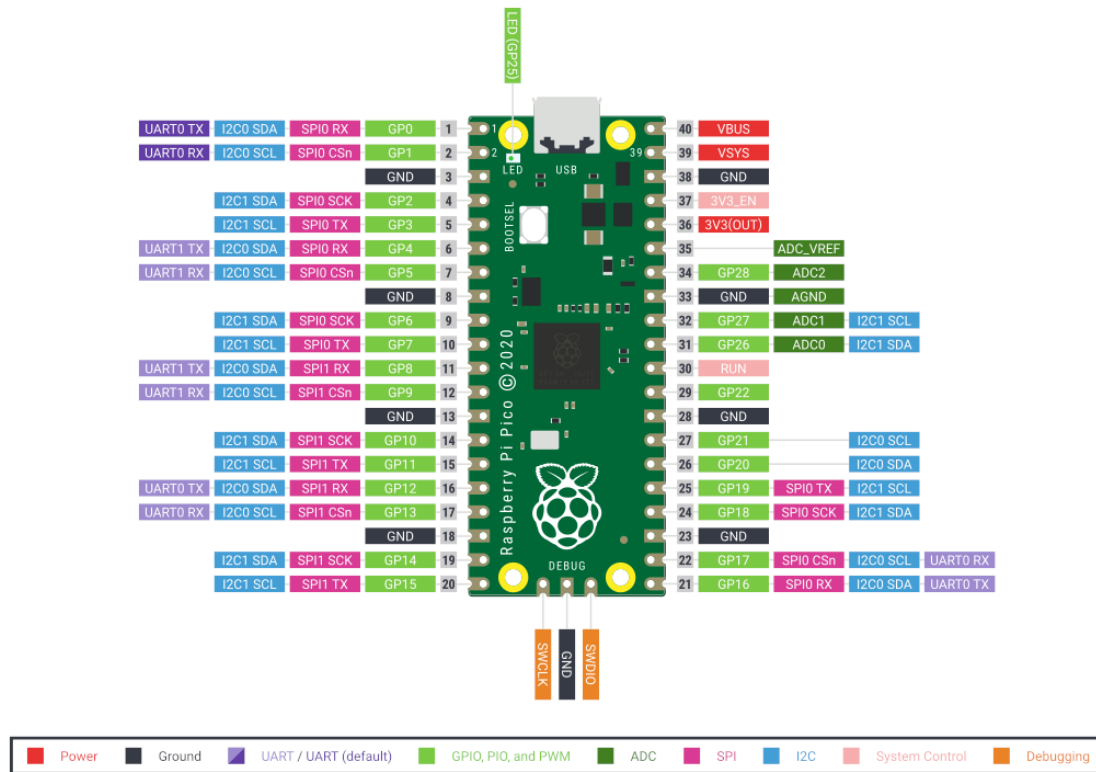
Figure 18. 7-segment digit states for all digits from 0 to 9

For this project, the Raspberry Pi Pico MCU is connected to the HDSP-523E. The digital pins of the Pico are not directly connected to the display but rather are connected through 330Ω resistors. As we do not need the dots for this project, they are directly connected to the ground.

The Raspberry Pi Pico MCU has 30 GPIOs (General Purpose Input/Output pins) labelled GP0 to GP29, of which 26 GPIO pins can be used for controlling the LEDs of the HDSP-523E. The pinout of the Pico is shown in Figure 19.

¹³ Lab 2 – Microcontroller and Output Display, Dr Ian Laird, 2023

¹⁴ Lab 2 – Microcontroller and Output Display, Dr Ian Laird, 2023



15

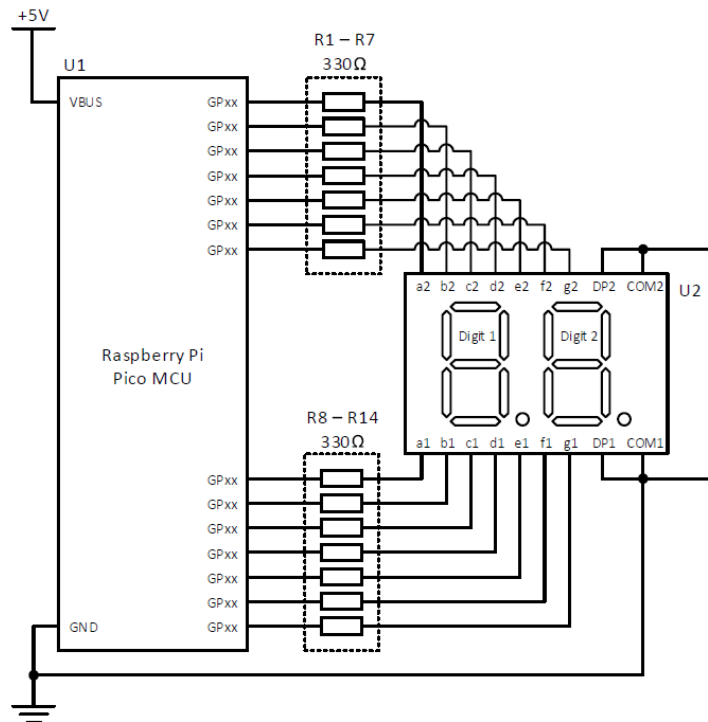
Figure 19. Raspberry Pi Pico pinout

Assemble the output hardware:

1. Put the HDSP-523E display onto the breadboard.
2. Put the Raspberry Pi Pico MCU onto the breadboard.
3. Connect the cathode pins and the dot pins to ground.
4. Connect all other pins of the HDSP-523E to the GPIO pins of the Pi Pico MCU with a 330 Ohm resistor in series in every connection.

The schematics of the assembled circuit should look like this:

¹⁵ Lab 2 – Microcontroller and Output Display, Dr Ian Laird, 2023



16

Figure 20. Circuit of the HDSP-523E display connected to the Raspberry Pi Pico microcontroller

To write, debug, and run a program on the Raspberry Pi Pico MCU, an IDE (Integrated Development Environment) is needed. The Pico has been developed for use with the Thonny IDE with MicroPython interpreter. The code for this lab has been written and run through the Thonny IDE.

We need to configure the used pins to display digits on the HDSP-523E dual-digit 7-segment. The pins can be configured as either input or output. For this, the "Pin" class of the "machine" module is needed.

When a pin is defined as an input, the "Pin" class gives the option to use either the pin's internal pull-up or pull-down resistor. Similarly, when a pin is defined as an output, the "Pin" class gives the option of setting the initial value of the pin as '0' (low; 0V) or as '1' (high; +3.3V). After initializing the pins, by using the "Pin" class "value" function, their values can be changed to the predetermined states ('low' or 'high').

After configuring the output pins, the pins must be grouped so the display can show digits. The pins can be grouped using sequences. The pins are grouped into separate sequences. There are two digit positions on the display used in this project, so two segments must be defined.

Table 3. Digit segments

Segment » Digit	A	B	C	D	E	F	G
First	pinA1	pinB1	pinC1	pinD1	pinE1	pinF1	pinG1
Second	pinA2	pinB2	pinC2	pinD2	pinE2	pinF2	pinG2

After configuring the sequences, the next step is to display the digits on the screen. There are 10 digits, and each one of them requires different segments to be activated. To achieve this, we create a table that contains values for each of the seven segments for all 10 digits. As explained in the previous section, the values '0' (off) and '1' (on) determine whether a segment is turned on or not. The truth table is shown in Table 4:

Table 4. Truth table representing the digits

State	A	B	C	D	E	F	G
0	ON	ON	ON	ON	ON	ON	OFF
1	OFF	ON	ON	OFF	OFF	OFF	OFF
2	ON	ON	OFF	ON	ON	OFF	ON
3	ON	ON	ON	ON	OFF	OFF	ON
4	OFF	ON	ON	OFF	OFF	ON	ON
5	ON	OFF	ON	ON	OFF	ON	ON
6	ON	OFF	ON	ON	ON	ON	ON
7	ON	ON	ON	OFF	OFF	OFF	OFF
8	ON	ON	ON	ON	ON	ON	ON
9	ON	ON	ON	ON	OFF	ON	ON

After being able to control every single segment and knowing its position on the display, the next step is to display the digits. For this reason, the program needs a function that, for any given digit, activates the corresponding segments to display the digit. The function assigns the

segment values for a digit, which are defined in the 'digits' table. This function can be named setDigitSegments.

Next, the program needs a function that is capable of displaying any number we want. To achieve this, we need to divide the number into separate digits and display them in their appropriate positions (i.e., 'ones', 'tens', 'hundreds'). This is done by using the modulus operator '%' to extract the digit, and then removing it using the operator '//'. However, we are using a dual-digit display in this project, so we only display the last two digits.

Now, we need to create a simple running timer program that displays numbers incremented after a set amount of time. To display the numbers, we use the setDisplayDigits function. For the delays, the 'sleep' function is imported from the 'utime' MicroPython module. As the 'int' data type can only store 8-bit numbers (the maximum decimal number is 256), to prevent overflow, the counter should be reset. To prevent numbers less than x10 (where x is any number greater than or equal to 1) but higher than x00 from being shown as single digits without a leading zero, the counter is reset to 11 whenever it reaches 111.

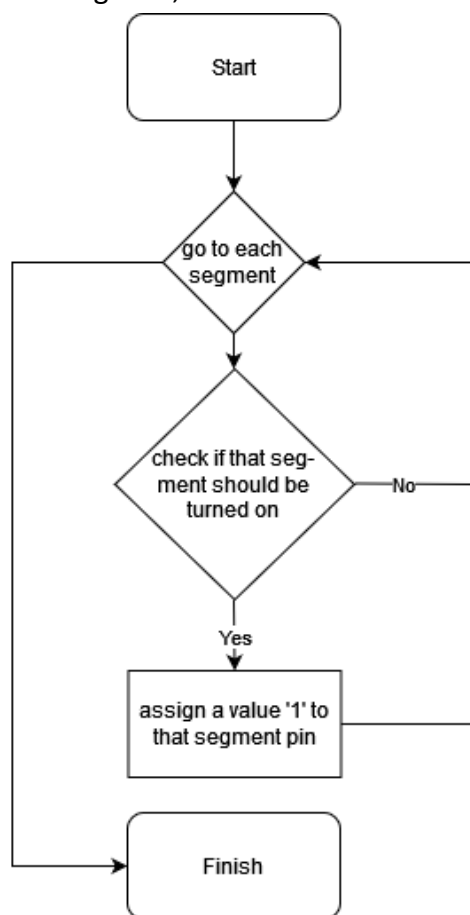


Figure 21. setDigitSegments function flowchart

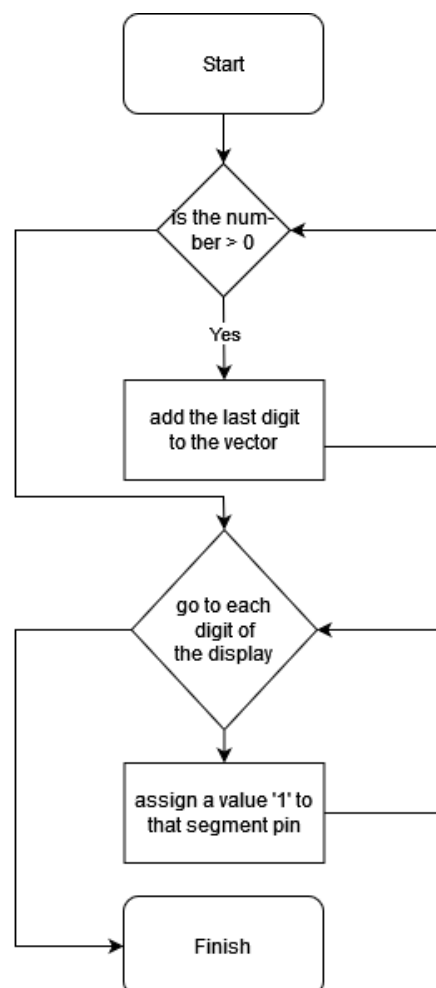


Figure 22. setDisplayDigits function flowchart

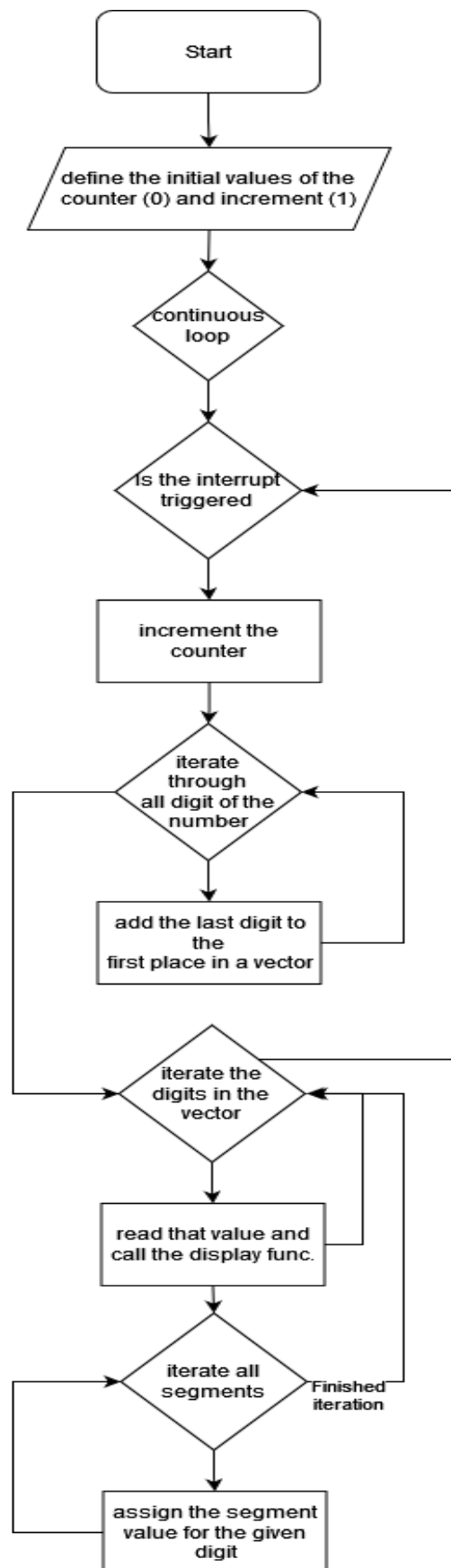


Figure 23. Full program flowchart

21/02/2023

Experimental Procedure and Results

After assembling the output hardware, to check whether all of the connections are working properly, we can turn the output of the pins to 1 (digital high, i.e., +3.3 V). To do this, we can simply iterate through all of the pins with the following code:

```
for i in range(27):  
    Pin(i, Pin.OUT, value = 1)
```

Figure 24. Code snippet of the loop that turns high output at every single pin

To do this, the following steps should be followed:

1. Connect the Raspberry Pi Pico microcontroller to a PC via a micro-USB cable.
2. Open the above-mentioned code in Thonny (IDE).
3. Debug and run it on the MCU.

During testing, all of the pins turned on, which means that all connections have been assembled correctly.

As the output hardware is working as intended, we can continue by running the final program. We need to define the specific output pins, group them in sequences, and create possible output combinations. Using the instructions in the Design Process, they are configured as shown in the following code snippet:

```

pinA1 = Pin(15, Pin.OUT, value=0) #a1
pinB1 = Pin(19, Pin.OUT, value=0) #b1
pinC1 = Pin(1, Pin.OUT, value=0) #c1
pinD1 = Pin(0, Pin.OUT, value=0) #d1
pinE1 = Pin(22, Pin.OUT, value=0) #e1
pinF1 = Pin(21, Pin.OUT, value=0) #f1
pinG1 = Pin(20, Pin.OUT, value=0) #g1

pinA2 = Pin(16, Pin.OUT, value=0) #a2
pinB2 = Pin(18, Pin.OUT, value=0) #b2
pinC2 = Pin(27, Pin.OUT, value=0) #c2
pinD2 = Pin(13, Pin.OUT, value=0) #d2
pinE2 = Pin(14, Pin.OUT, value=0) #e2
pinF2 = Pin(17, Pin.OUT, value=0) #f2
pinG2 = Pin(26, Pin.OUT, value=0) #g2

pins = [
    [pinA1, pinB1, pinC1, pinD1, pinE1, pinF1, pinG1] , # segments of the first digit
    [pinA2, pinB2, pinC2, pinD2, pinE2, pinF2, pinG2]  # segments of the second digit
]

digits = [
    [1, 1, 1, 1, 1, 1, 0] , # power truth values for 0
    [0, 1, 1, 0, 0, 0, 0] , # power truth values for 1
    [1, 1, 0, 1, 1, 0, 1] , # power truth values for 2
    [1, 1, 1, 1, 0, 0, 1] , # power truth values for 3
    [0, 1, 1, 0, 0, 1, 1] , # power truth values for 4
    [1, 0, 1, 1, 0, 1, 1] , # power truth values for 5
    [1, 0, 1, 1, 1, 1, 1] , # power truth values for 6
    [1, 1, 1, 0, 0, 0, 0] , # power truth values for 7
    [1, 1, 1, 1, 1, 1, 1] , # power truth values for 8
    [1, 1, 1, 1, 0, 1, 1]  # power truth values for 9
]

```

Figure 25. Code snippet of the pin configuration

To test if the correct pins are configured, in the above code, the values for the pins are temporarily set to 1 so they can all be turned on. Similarly, to the first test, we check the outputs.

This test showed that the correct pins had been configured.

Next, the function that turns on a specific combination of digits should be written and tested. The `setDigitSegments` function is written as explained in the Design Process. It is shown in the following code snippet:

```

def setDigitSegments(position, digit): # requires two inputs, position and digit
    for i in range(0, 7):             # iterates all seven segments
        pins[position][i].value(digits[digit][i]) # assigns a value to every segment

```

Figure 26. `setDigitSegments` code snippet

As the program can turn on the segments and display a number, a function that displays full numbers should be written and tested. The setDisplayDigits function is written as explained in the Design Process. It is shown in the following code snippet:

```
def setDisplayDigits(value):
    digits = []      # an empty vector in which the digits are kept
    dig = 0          # innitial number of digits (0)
    n = 2            # defines the number of used display digits (2)

    while value > 0:
        digits = [value % 10] + digits    # adds the last digit to the vector
        value //= 10                     # that digit is removed form the number
        dig += 1                          # number of stored digits is incremented by 1

    if dig > 1:
        dig -= 1
        for i in reversed(range(n)):
            setDigitSegments(i, digits[dig - i])    # displays the digit
    else: setDigitSegments(1, digits[0])            # if the number is 0
```

Figure 27. setDisplayDigits code snippet

The setDigitSegments and the setDisplayDigits functions were checked following these steps:

1. Display the following inputs with the setDigitSegments function: [[1, 3], [2, 3], [0, 7], [1, x], [1, aa], [0, 32]].
2. Display the following strings/numbers with the setDisplayDigits function: ['3', '8857', 'do5', 'd', ''].

In the tables below, the results acquired from the experimental testing are shown:

Test No.	Input	Output	Success?
1	[1, 3]	left digit displays 3	Yes
2	[2, 3]	error	?
3	[1, x]	error	?
4	[0, 7]	right digit displays 7	Yes
5	[1, aa]	error	?
6	[0, 32]	error	?

Test No.	Input	Output	Success?
1	'3'	3	Yes
2	'8857'	57	Yes
3	'do5'	error	?
4	'd'	error	Yes
5	"	nothing	Yes

The taken measurements confirmed that the code is running as intended.

The final task of the lab was to create a running timer. The code of the loop is shown in the following snippet:

```
i = 0
while True:
    setDisplayDigits(i) # displays the number
    utime.sleep(1)      # delays the next output by 1 second
    i += 1              # increments the
    if i > 110: i = 11  # prevents overflow
```

Figure 28. Timer code snippet

The final testing, now of the running timer is done to check whether the program has been written as intended. The testing confirmed that the code has been written and works as intended.

Conclusions and Recommendations

Experimental results demonstrated that:

1. The MCU and output display were successfully assembled and tested.
2. The program only accepts numerical values as input.
3. The program cannot process floating-point numbers.
4. The timer program functions correctly.
5. Functions setDigitSegments and setDisplayDigits work as intended, displaying only 1- or 2-digit numbers and only the last 2 digits of the number in case of overflow.

Recommendations and improvements for the work done include:

1. Add a feature to output on the display the distance between the proximity sensor and the detected object.
2. Incorporate a feature to output on the display the number of seconds the object was sensed by the proximity sensor.
3. Modify the program to produce an audible signal when the proximity sensor detects an object.

Lab 3 – Interface Electronics

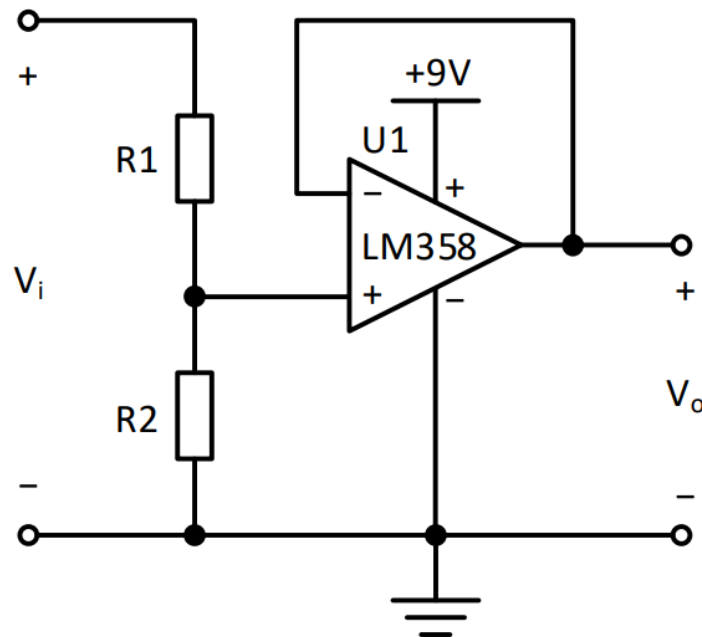
07/03/2023

Aims

- Develop a signal buffer to pass the lamp-post driver's output into the Raspberry Pi Pico microcontroller unit (MCU) while maintaining the voltage within the safe range from 2.4V to 3.3V.
- Create an interrupt service routine (ISR) for the MCU that counts the number of positive signals received from the lamp-post driver.
- Construct a Schmitt trigger that filters out unwanted noise that may occur when the lamp-post driver output signal transitions between high and low states.

Design Procedure

The signal buffer is used in the lab to convert the 0-9V output signal of the threshold comparator to a 0-3.3V signal that can be safely fed into the Raspberry Pi Pico MCU. The voltage divider is used to scale down the output of the threshold comparator to the desired voltage range, while the voltage follower ensures that the output of the voltage divider is not affected by the loading of the circuit that is connected to it, as shown in Figure 29.



17

Figure 29. Input signal buffer circuit

To create a voltage divider, it was necessary to choose appropriate values for the resistors R_1 and R_2 . The resistance values were selected using the Voltage Divider Rule and the ratio of the two resistors (k).

$$V_+ = \frac{R_2}{R_1 + R_2} V_i \quad (1)$$

Where V_+ - voltage supplied to the non-inverting input of the op-amp, V_i – output voltage of the threshold comparator, R_1 and R_2 - resistance of R_1 , and R_2 in Figure 29.

$$k = \frac{R_2}{R_1} \quad (2)$$

Using the ratio of the resistors (2), R_2 was expressed through R_1 .

$$R_2 = kR_1 \quad (3)$$

It is important to note that the actual values of the resistors do not define the voltage fed to the MCU; instead, their ratio does. Therefore, it was necessary to express the ratio in terms of one of the resistors, which in this case was R_1 . Thus, R_2 was replaced by kR_1 (3) in equation (1).

$$\begin{aligned} V_+ &= \frac{kR_1}{R_1 + kR_1} V_i \\ V_+ R_1 + V_+ kR_1 &= kR_1 V_i \\ V_+ R_1 &= kR_1 V_i - kV_+ R_1 \\ V_+ R_1 &= k(R_1 V_i - V_+ R_1) \\ k &= \frac{V_+ R_1}{(R_1 V_i - V_+ R_1)} \end{aligned} \quad (4)$$

Once k was expressed in terms of R_1 , known values were substituted into equation (4). Since R_1 is an independent variable and can have any value, it was set to be 1kΩ. Moreover, the actual value of V_i was found to be less than 9V, as there is a voltage drop across the components of the circuit that were before the op-amp, and it was measured to be approximately 7.5V. As for V_+ , the highest allowable voltage for the MCU was utilized. After deducing a formula for the ratio, R_2 was obtained by substituting R_1 and k into equation (4).

$$k = \frac{3.3 \times 1000}{(1000 \times 7.5 - 3.3 \times 1000)} = 0.786 \text{ (3 sig. fig.)}$$

As k was calculated, it was substituted, along with R_1 into (3) to find R_2 .

$$R_2 = 0.786 \times 1000 = 786\Omega \text{ (3 sig. fig.)}$$

To reduce the output voltage of the comparator from 7.5V to 3.3V, it was necessary to employ two resistors with resistance values of 786Ω and 1000Ω. It should be noted that the smaller the value of R_2 , the lower the voltage will be. Given this fact, and because a 786Ω resistor was not available in the lab kit, a 650Ω resistor was chosen as a substitute. Substituting this value into (1) results in a voltage of 2.95V, which falls within the safe range.

$$V_+ = \frac{650}{650 + 1000} \times 7.5 = 2.95V \text{ (3 sig. fig.)}$$

After assembling the signal buffer, we had a circuit that detected movement and generated an appropriate signal. To be able to count, the program needed to check the GPIO pin (connected to the Signal Buffer) and see if there are any voltage changes from the digital low state (0 V) to the digital high state (approx. +3.3V). If a rising signal is detected, that means an object has been moved into the range of the proximity sensor, and the counter value should be incremented. For that reason, the program should continuously observe the defined input GPIO pin for a rising edge.

There were two possible ways to do this:

1. Placing an if statement within an infinite loop that constantly monitors the voltage of the pin and only increments the counter if a rising edge has occurred. This is also called "polling" a pin.
2. Using hardware interrupt that monitors the pin for a rising edge and increments the counter when it detects one in an interrupt service environment (ISR).

Polling a pin can limit a program's ability to perform other tasks, especially as the program becomes more complex. In contrast, the interrupt method delegates pin monitoring to a dedicated hardware module, freeing up the program to perform other operations. Once an interrupt occurs, the program completes its current operation and hands control to an Interrupt Service Routine (ISR), which functions like a typical function. Interrupts have the advantage of allowing a program to perform multiple tasks without having to constantly poll pins.

To complete the system's software program, the following steps were required:

1. Configure the GPIO pin connected to the output of the signal buffer as an input.
2. Attach an interrupt to the same GPIO pin and configure it so that it is triggered by a rising edge on the pin.
3. Create an ISR that increments a counter and displays the result on the dual-digit, 7-segment display.

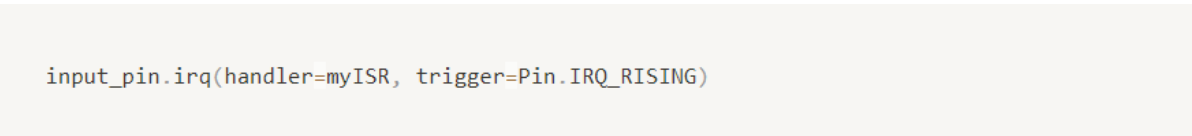
The input GPIO pin was defined with the following line of code:



```
input_pin = Pin(3, Pin.IN, Pin.PULL_DOWN)
```

Figure 30. Input GPIO code snippet

Adding an interrupt to the defined GPIO pin was done with the following line:



```
input_pin.irq(handler=myISR, trigger=Pin.IRQ_RISING)
```

Figure 31. ISR code snippet

The ISR was defined similarly to any other function; however, it had some distinct requirements. ISRs differ from other functions as they can only have one input argument and cannot return any variables. This means that variables cannot be passed into or returned from an ISR, and all declared variables are only in scope within the ISR. However, global variables can be used to manipulate variables used outside the ISR. To achieve this, the variable must be declared global within the ISR before its use. Only altered variables that must retain their new value outside the ISR must be declared global.

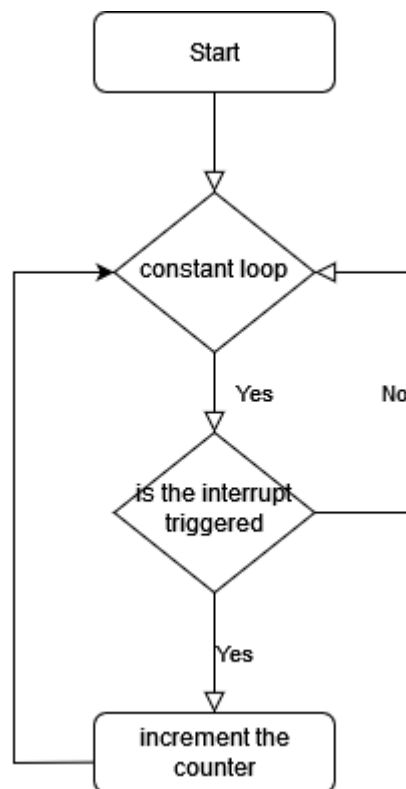
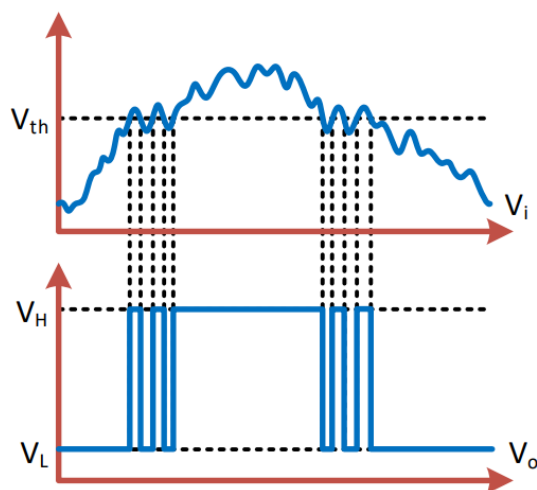


Figure 32. ISR flowchart

The interrupt function, as well as the adjusted timer, were added to the code written in Lab 2, and the final program is created.

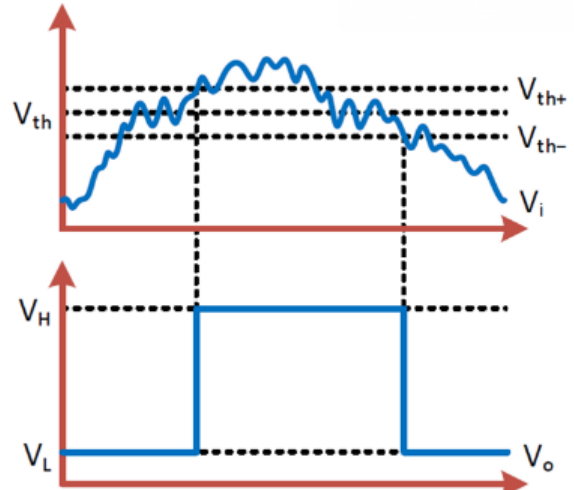
To address the issue of electrical noise causing the number on the output display to increment by more than a single value as intended, a Schmitt trigger was utilized. As depicted in Figure 35, the Schmitt trigger is composed of two resistors, R_2 and R_3 , which define the circuit's sensitivity to signal fluctuations and noise.

The following two waveforms show the tolerance level that the Schmitt trigger provides.



18

Figure 33. Threshold comparator

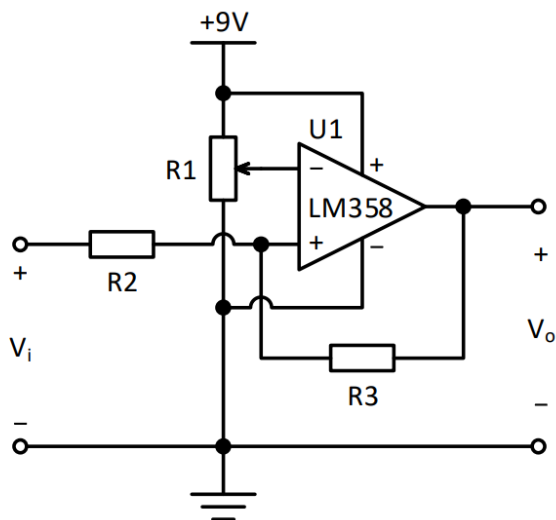


19

Figure 34. Schmitt trigger

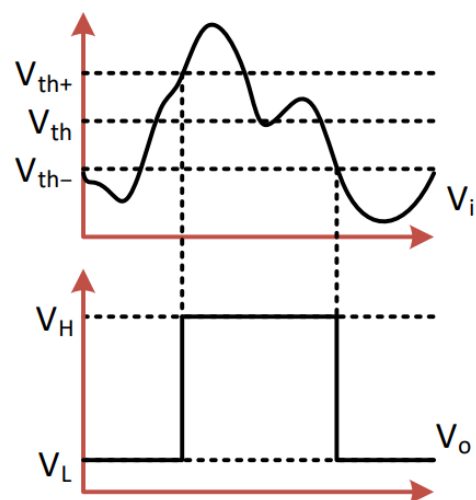
$$V_{th+} = V_{th} + \frac{R_A}{R_b} (V_{th} - V_L)$$

$$V_{th-} = V_{th} - \frac{R_A}{R_b} (V_H - V_{th})$$



20

Figure 35. Non-inverting Schmitt trigger circuit schematic



21

Figure 36. Operational waveforms

¹⁸ Lab 3 – Interface Electronics, Dr Ian Laird, 2023

¹⁹ Lab 3 – Interface Electronics, Dr Ian Laird, 2023

²⁰ Lab 3 – Interface Electronics, Dr Ian Laird, 2023

²¹ Lab 3 – Interface Electronics, Dr Ian Laird, 2023

14/03/2023

Experimental Procedure and Results

To confirm the functionality of the signal buffer, the following steps were necessary:

1. Construct the voltage divider using the calculated resistors as depicted in Figure 29.
2. Connect the pins of the LM358 according to the circuit diagram shown in Figure 35. Refer to Figure 9 for the position of the pins.
3. Connect one pin of the potentiometer to the output pin of the LM358.
4. Connect another pin of the potentiometer to the ground (GND).
5. Power on the circuit and record the voltage on the potentiometer.

After constructing the signal buffer, it was verified that the output voltage remained within a safe range and was equal to roughly 2.9V. The output of the signal buffer was then connected to an unused GPIO pin on the Raspberry Pi Pico MCU.

To determine whether the Schmitt trigger functions as intended by filtering out noise from the signal, it must be assembled. Through trial and error, it was determined that the counter ceased responding to electrical noise at values of 33k Ω and 1M Ω , respectively. The Schmitt trigger is constructed by placing the two resistors on the threshold comparator.

Finally, the program was tested with the completed circuit, and it was concluded that the circuit operates as intended, and that the project objectives were achieved.

Conclusions and Recommendations

The experimental results demonstrate the following findings:

- The voltage supplied to the MCU was equal to 2.9V, which falls within the safe range for this type of microcontroller.
- The ISR functioned as intended, enabling the program to track the number of objects detected by the proximity sensor and display the output accordingly.
- The circuit's operation remains unaffected by electrical noise, and the display does not increment by multiple values when an object is detected.
- The display is limited to showing numbers of two digits or less.
- The display increases by one even if multiple objects pass through the transmitter's IR radiation simultaneously or near each other.

Potential improvements for this study include:

- Upgrade to a larger display showing numbers with three or more digits.
- Modify the proximity sensor and reprogram the circuit to differentiate between a single object passing through the IR radiation and multiple objects passing through concurrently.

- Implement a feature that displays the speed of a moving object by installing an additional sensor at a specific distance from the first sensor and rewriting the existing program accordingly.

Finally, the entire circuit looked like this in the picture provided below:

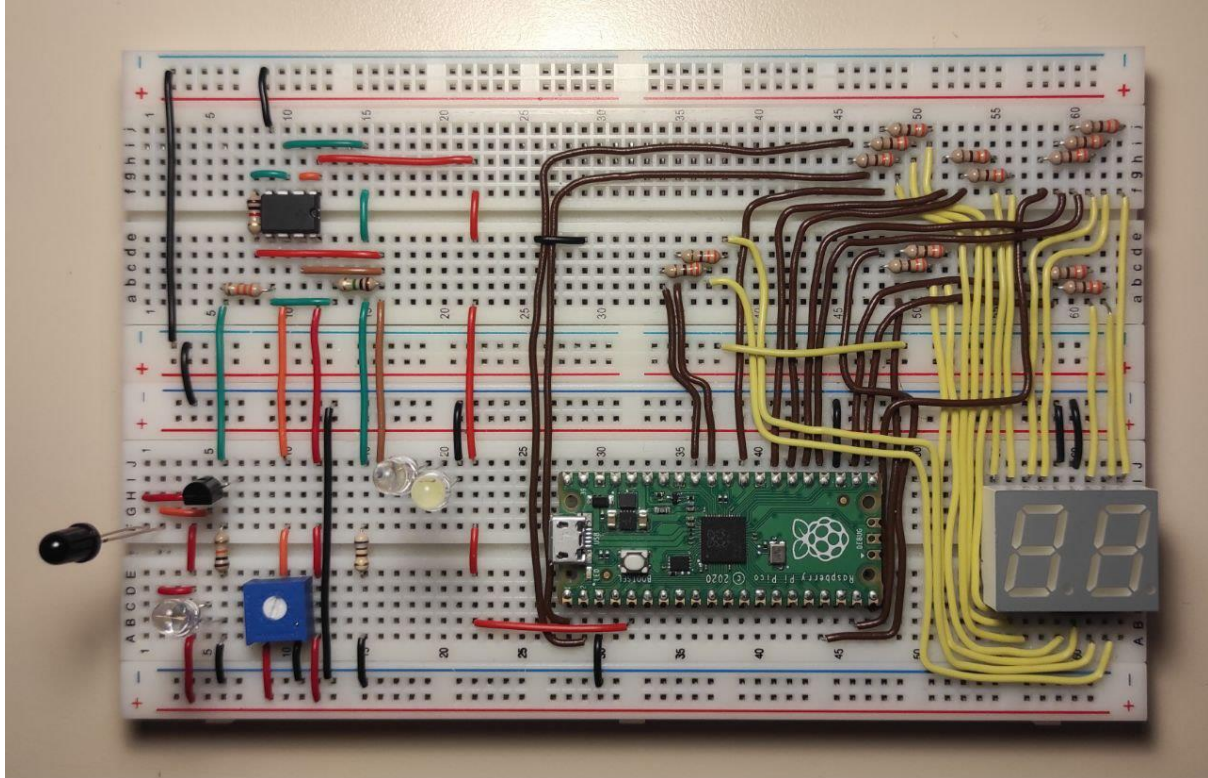


Figure 37. The final circuit of the project