HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY – VNU HCMC
OFFICE FOR INTERNATIONAL STUDY PROGRAM
FACULTY OF ELECTRICAL AND ELECTRONIC ENGINEERING

———————— * ————————

COMPUTER SYSTEMS
PROJECT REPORT
Design a product counter using an ARM processor

Lecturer : **Dr. Trương Quang Vinh**
Subject : **Computer Systems**
Class : **TT01**
Group : **10**
Member : **Lương Triển Thắng - 2051194**
Đinh Hoàng Luân - 2051145
Phan Quang Minh - 2051052
Bùi Thanh Tùng - 2051213

Ho Chi Minh City, 12th December, 2022

Contents

I Abstract

In this project, our team will create (hardware and software) to demonstrate the product counter by using ARM processor (ARM Cortex M0+ processor), particularly, Raspberry Pi Pico microcontroller along with some other components: infrared proximity Sensor for detecting objects, four-digit 7-segment display for displaying product count, a buzzer for creating a sound when the an object is counted, and 3D printed case for protecting the devices. There are two methods: Program Loop and Interrupt Vector. Our team choose program loop method, the MCU will be constantly in a loop for reading IR, checking the IR state. So the program uses all CPU performance just for this simple task. Finally, we will give the experimental result and oriented development in the near future.

II Introduction

1. Introduce the problem

In the current technological era, with the strong development of science and technology, product counting models were born based on chip manufacturing technology. Counters are used to keep track of the number of products produced in a production line or even in a factory. In addition, the counter is also used for data acquisition, remote monitoring, automation system. In this project, our team will describe the objectives of product counter by using an ARM processor, as you know, the product counter is most widely used in the production of quantities of lines, for example:

- Pharmaceutical products
- Manufacturing applications
- Factory applications
- Food and beverage counting
- Can counting
- Part and component counting and batching

To be more specific:

- Product or Manage Vehicle Number Out-In:
  The infrared sensor will receive a signal when there is an obstacle and count the number displayed on the 7-segment LED.
- Cement bag counter
  The set of equipment includes: LED display panel and product counting sensor. The LED display panel will be installed in a position so that the manager can easily observe. Product counting sensor is installed at the conveyor belt.

ARM processor is one of a family of central processing units based on the reduced instruction set computer architecture for computer processors. There have been several generations of the ARM design, including the ARM Cortex-M processor family is based on the M-Profile Architecture that provides low-latency and a highly deterministic operation for deeply embedded systems. From the above, our team decided to choose ARM Cortex M0+ Processor to implement the project, specifically Raspberry Pi Pico microcontroller chip, because of the following advantages that it brings:

- Firstly, the Cortex-M0+ processor has the smallest footprint and lowest power requirements of all the Cortex-M processors. This is well-suited for low-cost devices, including smart sensors and mixed-signal systems on chip, adding intelligence to devices that were not capable before.
- Additionally. the exceptional code density of Cortex-M0+ significantly reduces memory requirements, which maximizes the use of on-chip Flash memory to save memory cost, reduce memory power, and increase maximum performance. Take advantage of 32-bit processing intelligence at an 8/16-bit processor cost point.
- Finally, the Cortex-M0+ processor allows developers to optimize power usage for specific ap-

plications with built-in, low-power features. With its three highly optimized low-power modes, the processor conserves energy to match processing demands.

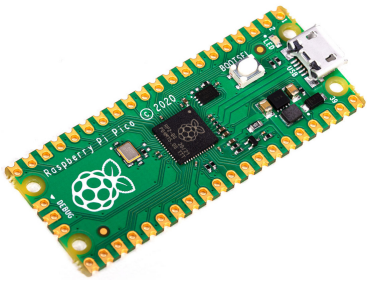
In this project, we will use Raspberry Pi Pico. It is a MCU (Micro Controller Unit), an integrated circuit on a programmable chip set used to control the operation of the system. The microcontroller reads, stores information, processes information, measures time, and interfacing with IO devices. The programmer can use many languages to program the microcontroller. But the commonly used languages are C and Assembly. The MCU acts as the "brain" that governs all the behavior of the circuit board. Therefore you will have to program it so that it does the job you want. With a lot of outstanding power, it is extremely simple to use the Raspberry Pi Pico microcontroller chip to build a product counter. In addition to the small cost, It also offers a high processing speed and wide customization capabilities that are very suitable for counter design.

2. Describe the objectives

We expected a real working product counter device by the end of the project.

The device consists of:

- Raspberry Pi Pico (ARM Cortex M0+): the "brain" of the device.
- Infrared Proximity Sensor, with built-in sensitivity control potentiometer: for detecting objects.
- Four-digit 7-segment display, with built-in shift register: display product count.
- 3D printed case/housing.



Raspberry Pi Pico

Proximity Sensor

Four-digit 7-segment display

The software would be written in C, using Keil-C and Pico device library[1].

All resources will be open-sourced and uploaded on our GitHub repository.

[1] Gabriel Wang. *Pico Template*. 2022. URL: `https://github.com/GorgonMeducer/Pico_Template`.

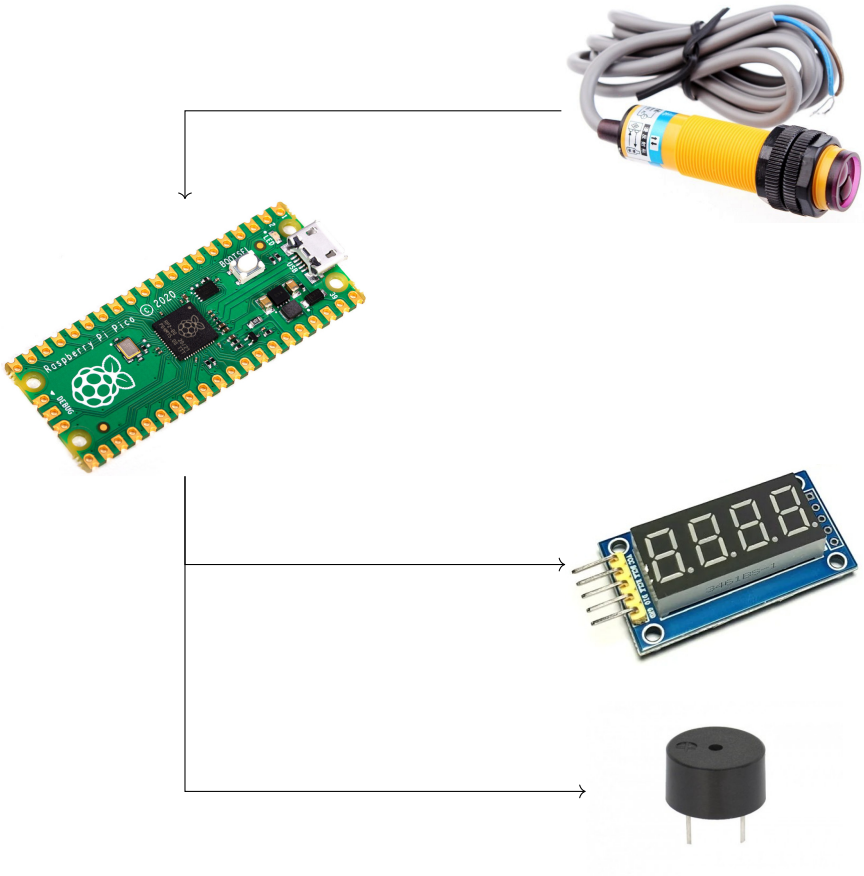
III Algorithm

1. Describe the algorithm

This device counts the number of obstacles that pass in front of the IR sensor in one direction only. The value of the total counts or the count number is displayed on a four-digit 7-segment display module. The IR sensor that we are using in this project is an IR proximity sensor. Whenever it detects an object inside its range the output generated by it is high otherwise the output is low. The count is zero initially and then incremented by one whenever something passes in front of it.

2. Flowchart

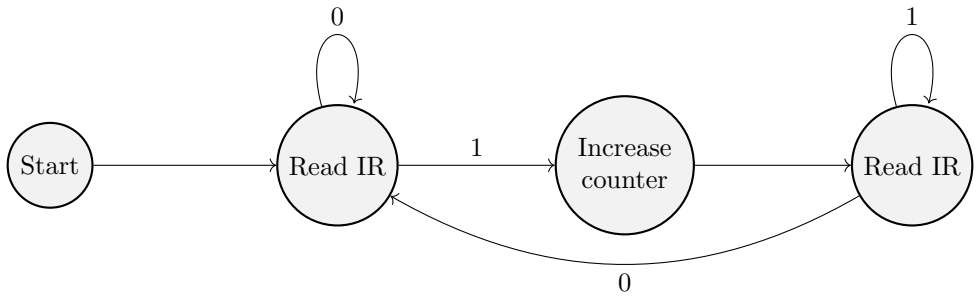
a. IO schematic



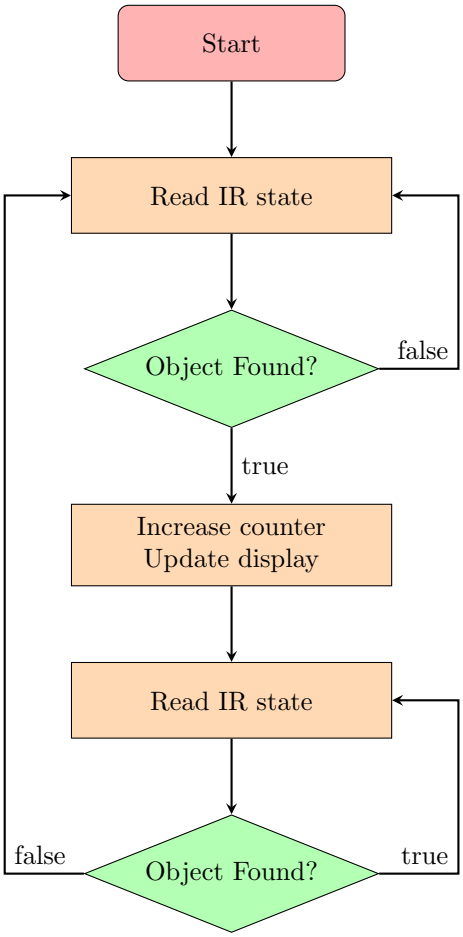
b. Method 1: program loop

Using this method, the MCU will be constantly in a loop for reading IR, checking the IR state. So the program uses all CPU performance just for this simple task, wasting all of the power of the ARM Cortex M0+.

Finite State Machine



Program Flow



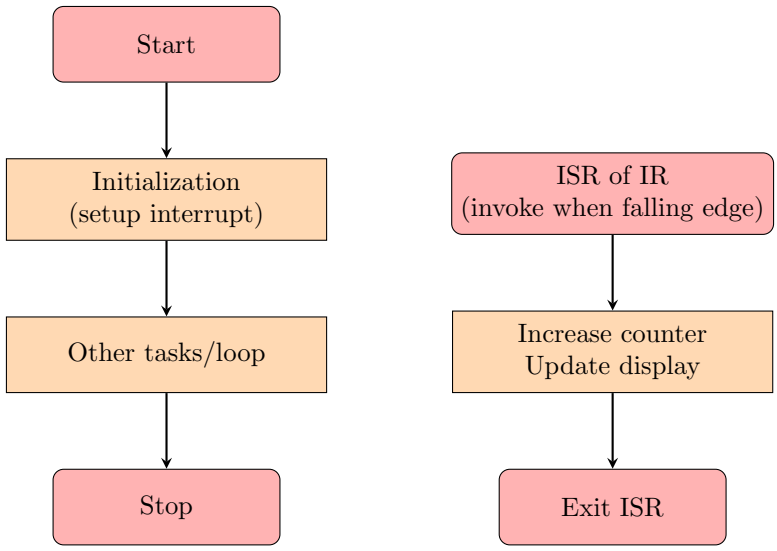
Pseudo Code

```
LOOP:
    COUNT := 0
    IR := ReadIR()
    IF (IR == HIGH) THEN
        COUNT++
        UPDATE 7-segment display
        IR = ReadIR()
        WHILE (IR == HIGH) DO
            IR = ReadIR()
        END WHILE
        GOTO LOOP
    END IF
    GOTO LOOP
```

c. Method 2: using interrupt

By using this method, the counter increment will just process when there is interruption of the proximity sensor, leaving all CPU free for other heavy tasks when there is no object detected.

Program Flow



(a) Main program (b) Sensor ISR

Pseudo Code

```
1  MAIN PROGRAM:
2      INIT_ISR
3      COUNT := 0
4      DO TASKS / LOOPS
5  END PROGRAM
6
7  IR_ISR:
8      COUNT++
9      UPDATE 7-segment display
10 END IR_ISR
```

IV Hardware

1. Raspberry Pi Pico

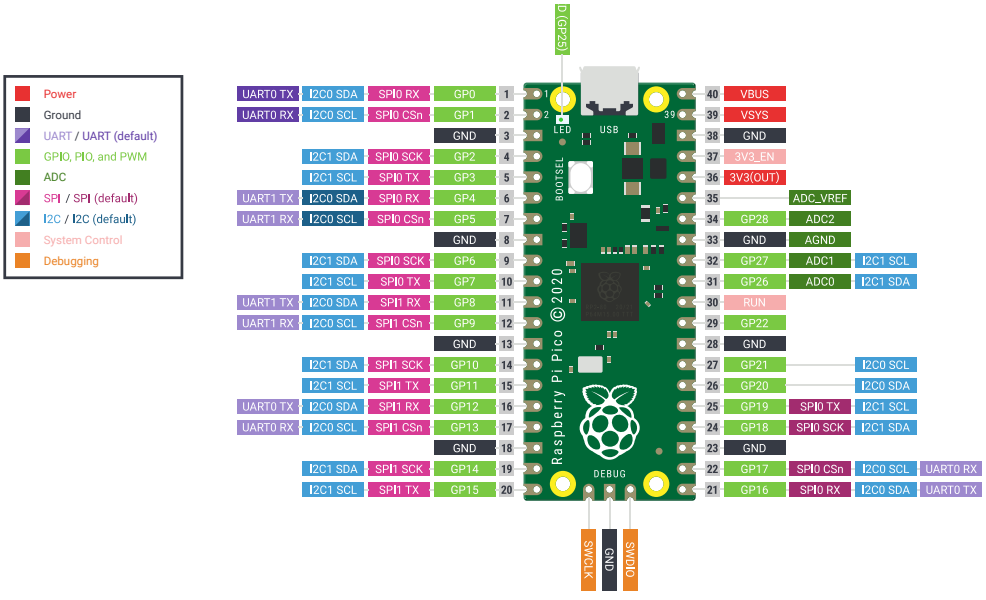
Raspberry Pi Pico is a low-cost, high-performance microcontroller board with flexible digital interfaces. It incorporates Raspberry Pi's own RP2040 microcontroller chip, with a dual-core ARM Cortex M0+ processor running up to 133 MHz, embedded 264KB of SRAM, and 2MB of onboard Flash memory, as well as 26 × multi-function GPIO pins.

For software development, either Raspberry Pi's C/C++ SDK or the MicroPython is available. In this project, we will use Raspberry Pi's C/C++ SDK, along with Gabriel Wang's Pico Template to use with Keil C.

Features

- RP2040 microcontroller chip designed by Raspberry Pi in the United Kingdom.
- Dual-core Arm Cortex M0+ processor, the flexible clock running up to 133 MHz.
- 264KB of SRAM, and 2MB of onboard Flash memory.
- USB 1.1 with device and host support.
- Low-power sleep and dormant modes.
- 26 × multi-function GPIO pins.
- 2 × SPI, 2 × I2C, 2 × UART, 3 × 12-bit ADC, 16 × controllable PWM channels.
- Accurate clock and timer on-chip.
- Temperature sensor.
- Accelerated floating-point libraries on-chip.
- 8 × Programmable I/O (PIO) state machines for custom peripheral support.

Pinout diagram



2. Infrared Proximity Sensor

An infrared sensor is an electronic device, that emits in order to sense some aspects of the surroundings. An IR sensor can measure the heat of an object as well as detects the motion. These types of sensors measure only infrared radiation, rather than emitting it that is called a passive IR sensor. Usually, in the infrared spectrum, all the objects radiate some form of thermal radiation.

These types of radiations are invisible to our eyes, which can be detected by an infrared sensor. The emitter is simply an IR LED (Light Emitting Diode) and the detector is simply an IR photodiode that is sensitive to IR light of the same wavelength as that emitted by the IR LED. When IR light falls on the photodiode, the resistances and the output voltages will change in proportion to the magnitude of the IR light received.

There are two IR LED in an IR sensor:

- Infrared Transmitter: an LED which emits infrared radiations. Even though an IR LED looks like a normal LED, the radiation emitted by it is invisible to the human eye.
- Infrared receivers or infrared sensors: detect the radiation from an IR transmitter. IR receivers come in the form of photodiodes and phototransistors. Infrared Photodiodes are different from normal photo diodes as they detect only infrared radiation.

Figure 4.1: IR transmitter (left) and IR receiver (right)

When the IR transmitter emits radiation, it reaches the object and some of the radiation reflects back to the IR receiver. Based on the intensity of the reception by the IR receiver, the output of the sensor defines.

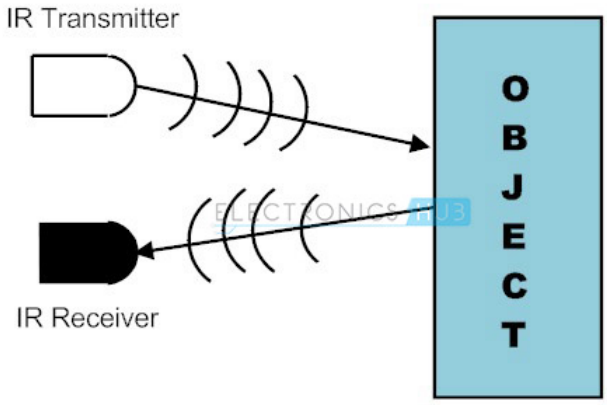


Figure 4.2: Infrared sensor working principle[1]

[1]Electronics Hub. *IR (Infrared) Obstacle Detection Sensor Circuit.* 2015. URL: `https://www.electronicshub.org/ir-sensor/`

In this project, we will be using an E3F-DS30C4 Adjustable IR Infrared Proximity Sensor.

Figure 4.3: E3F-DS30C4 Adjustable IR Infrared Proximity Sensor

E3F-DS30C4 Adjustable IR Infrared Proximity Sensor provides a good distance measuring (5 to 30 cm), power supply reversal connection protection, short-circuit protection, a small potentiometer to adjust sensitivity and more. But the most significant feature is signal interference prevention, best for detecting moving objects. There are three wires on the sensor:

- Brown: Power source (5 to 36 V)
- Blue: Ground
- Black: Signal output, the sensor is normally open (active low), so when the sensor detect objects, the output signal will be low.

3. Four-digit 7-segment display with built-in shift registers

a. 7-segment display

A seven-segment display is a form of electronic display device for displaying decimal numerals. As the name stated, the device uses seven segment (denoted from a to g) to display a number or a character in alphabet. Some device even contains a decimal point segment (denoted as p or dp). [14]

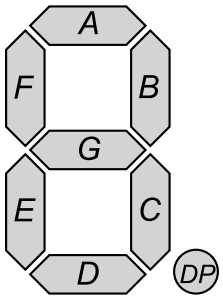


Figure 4.4: 7-segment display segments naming

Every segment is represented by one-bit value, 1/0 (active high) or 0/1 (active low) for on/off. Here is a table for binary and hexadecimal encoding for displaying 0 to F.

Table 1: Binary and hexadecimal encoding for displaying 0 to F

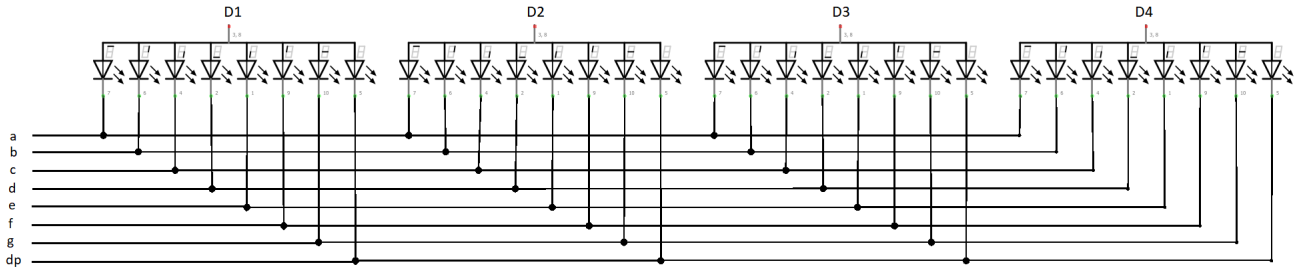
Digit	Display	p	a	b	c	d	e	f	g	abcdefg	hex pabcdefg	$\overline{\text{pgfedcba}}$	hex $\overline{\text{pabcdefg}}$
0	0	0	1	1	1	1	1	1	0	01111110	0x7E	11000000	0xC0
1	1	0	0	1	1	0	0	0	0	00110000	0x30	11111001	0xF9
2	2	0	1	1	0	1	1	0	1	01101101	0x6D	10100100	0xA4
3	3	0	1	1	1	1	0	0	1	01111001	0x79	10110001	0xB0
4	4	0	0	1	1	0	0	1	1	00110011	0x33	10011001	0x99
5	5	0	1	0	1	1	0	1	1	01011011	0x5B	10010010	0x92
6	6	0	1	0	1	1	1	1	1	01011111	0x5F	10000010	0x82
7	7	0	1	1	1	0	0	0	0	01110000	0x70	11111000	0xF8
8	8	0	1	1	1	1	1	1	1	01111111	0x7F	10000000	0x80
9	9	0	1	1	1	1	0	1	1	01111011	0x7B	10010000	0x90
A	A	0	1	1	1	0	1	1	1	01110111	0x77	10001000	0x8C
b	b	0	0	0	1	1	1	1	1	00011111	0x1F	10000011	0xBF
C	C	0	1	0	0	1	1	1	0	01001110	0x4E	11000110	0xC6
d	d	0	0	1	1	1	1	0	1	00111101	0x3D	10100001	0xA1
E	E	0	1	0	0	1	1	1	1	01001111	0x4F	10000110	0x86
F	F	0	1	0	0	0	1	1	1	01000111	0x47	10001110	0x8E
-	-	0	0	0	0	0	0	0	1	00000001	0x01	10111111	0xBF
off		0	0	0	0	0	0	0	0	00000000	0x00	11111111	0xFF

b. Multiplexed display

Multiplexed displays are basically multiple displays which are multiplexed together and usually one is turned on at a time, the turning on and off of multiple displays are so fast that viewer is able to believe that all the displays are turned on at a time. [4]

Multiplexing of displays have some benefits like reduced number of input pins, reduced complexity and less power is required for the display. In this project we will use 4-digit 7-segment multiplexed display. If we use 4 different one-digit 7-segment (with decimal point) display, we need $9 \times 4 = 36$ pins. On this multiplexed display, we only need $8 + 4 = 12$ pins only.

Front view and pinout



Schematic

Figure 4.5: Four-digit 7-segment display[1]

To turn on a segment, output high to desired the digit (`D1`, `D2`, `D3` or `D4`) and output low to the disired segment (`a`, `b`, `c`, `d`, `e`, `f`, `g`). The current flows from high potential to low potential to light up the segment.

Assume we want to display 1111, it is easy as we can give output high to `D1`, `D2`, `D3`, `D4` (4 columns to turn on all 4 digits) and low to b, c (segment b and c is to drive number 1 on a seven segment). But in case we want to display 1234, it can not be easy to do, if we drive 1 using segments then all the digits will display 1 since there is no separate segment pins for each individual digit.

Solution to this problem is making use of multiplexing, first output high to `D1` and output low to all other digits and display 1, then output high to `D2` and output low to all other digits and display 2, same goes for `D3` and `D4` and we start this over. When interval between turning high and turning low is few milliseconds, a human eye will not be able to see any blinking and viewer will feel all the digits are on at a time.

c. Shift register 74HC595

TM74HC595 is an open-drain output CMOS shift register which is designed with controllable tri-state output terminals and, when in serial output configuration, can control cascade chip of next stage. This product is excellent in performance and reliable in quality.

The 74HC595 is capable of handling high-speed shift clock frequency ($F_{\max} > 25\text{MHz}$), Standard SPI, CMOS serial output and capable of cascading multiple devices. This device can be treated as a serial-to-paralell data converter.

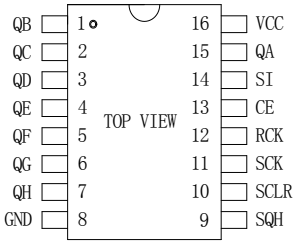


Figure 4.6: TM74HC595 Pinout[2]

The schematic below will explain how the IC works.

[1] Nikhil Joshi. *What are multiplexed displays and use of multiplexing - Arduino - C Programming,Seven Segment - Dotnetlovers.* 2020. URL: `https://www.dotnetlovers.com/article/10246/what-are-multiplexed-displays-and-use-of-multiplexing`

[2] *CMOS Shift Register TM74HC59.* Titan Micro Electronics

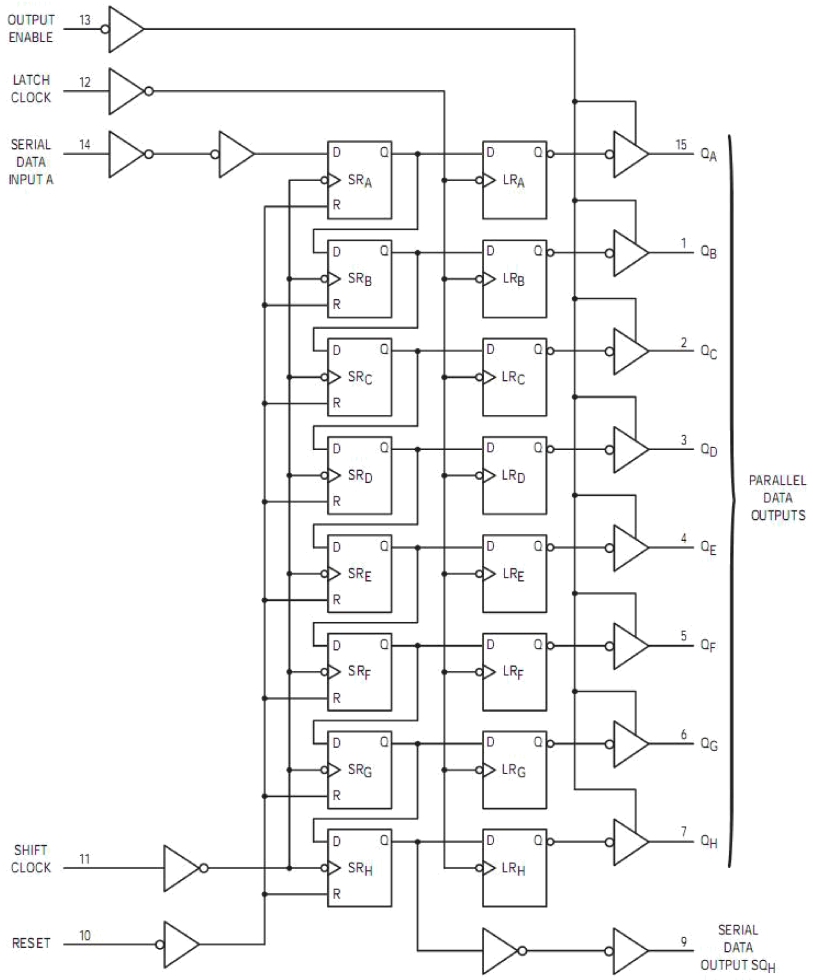
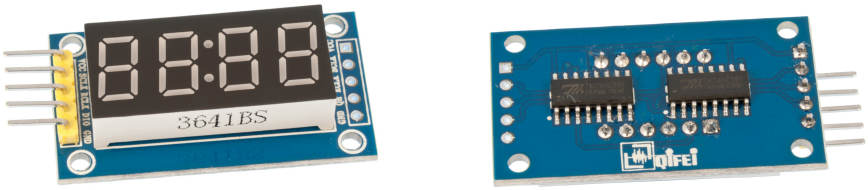


Figure 4.7: TM74HC595 Schematic[1]

d. Multiplexed Display with 74HC595



Front view Back view

Figure 4.8: Four-digit 7-segment display with 74HC595 module

[1] *CMOS Shift Register TM74HC59*. Titan Micro Electronics

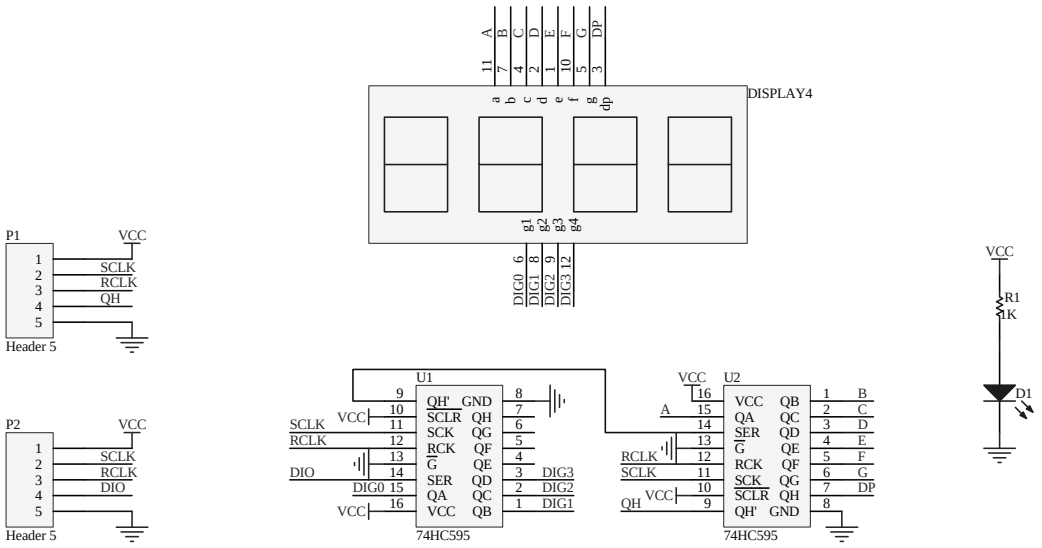


Figure 4.9: Four-digit 7-segment display with 74HC595 module schematic

Module pinout:

- `VCC`: Power source
- `SCLK`: Shift clock
- `RCLK`: Latch clock
- `DIO`: Serial data input
- `QH`: Serial data output
- `GND`: Ground

According to the schematic, the module contains two 74HC595s. The first one drives the signal for `D1`, `D2`, `D3` and `D4`. The second one drives the signal for the segments.

`SCLK` and `RCLK` will in turn provide the clock for the flip flop in the active 7-segment display. Specifically, `SCLK` helps to temporarily store the value of the number that needs to be released next, while `RCLK` will perform to update the display output of 7-segment display.

For example, to show 1234 on the display, according to the table above, the data is `F9 A4 B0 99 = 11111001 10100100 10110001 10011001`, respectively. Here is the data sequence need to be sent:

- Send the 1st digit data: 11111001.
- Send 1000 to enable `D1`.
- Send the 2nd digit data: 10100100.
- Send 1000 to enable `D2`.
- Send the 3rd digit data: 10110001.
- Send 1000 to enable `D3`.
- Send the 4th digit data: 10011001.
- Send 1000 to enable `D4`.

4. Buzzer

A buzzer or beeper is an audio signaling device, which may be mechanical, electromechanical, or piezoelectric. Typical uses of buzzers and beepers include alarm devices, timers, train and confirmation of user input such as a mouse click or keystroke.

A typical buzzer has two wire: anode (+) and cathode (−). The buzzer usually makes use of a material that has piezoelectric properties. When a material is piezoelectric, it means that applying mechanical force will cause it to gain charge. Inversely, applying an electric charge will cause the

material to stretch or expand a very small amount. By pulsing our buzzer on and off we make the piezoelectric material inside quickly expand and contract. This produces the vibrations we hear in the air. So, in order to make it works, apply 5V to anode and ground to cathode.

Figure 4.10: A buzzer

5. Wiring schematic

The schematic is drawn using Fritzing[1].

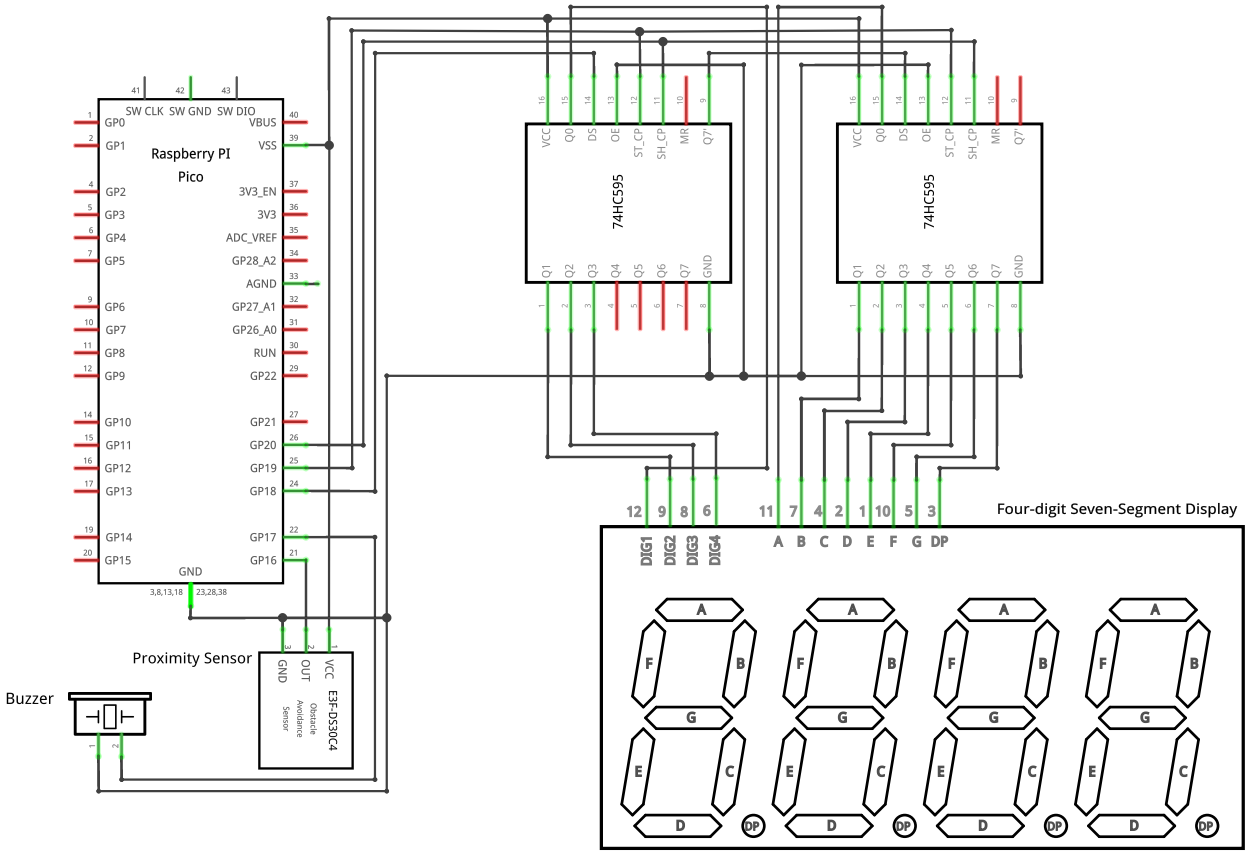


Figure 4.11: Overall wiring schematic

6. 3D Printed Case

To finish the hardware part, we use Autodesk Fusion 360 to design a 3D printed case for making the device more compact and portable. A rectangular hole at the top for four-digit 7-segment display. A circular and rectangular holes at two sides for proximity sensor and micro USB port. A detail CAD schematic with dimentions is presented in Appendix section.

[1]Fritzing is an open-source electronics design and prototyping platform for makers, hobbyists, and educators.

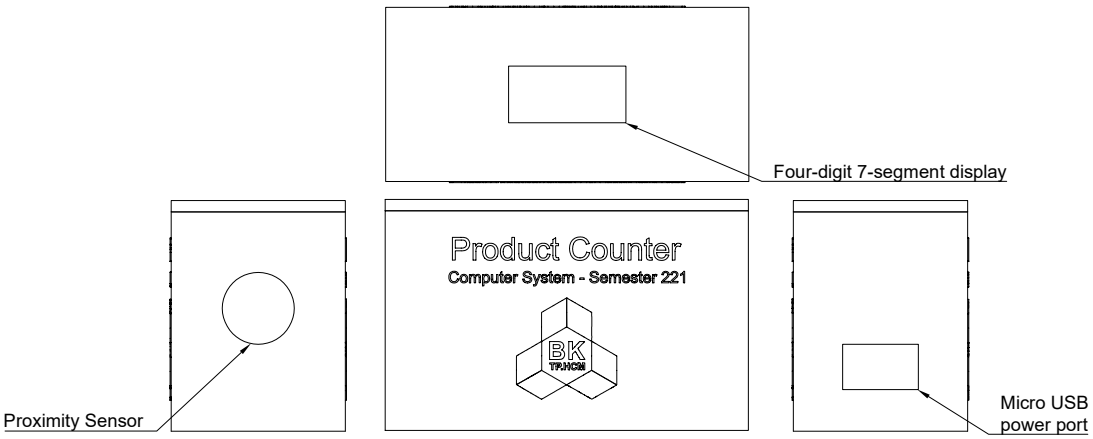


Figure 4.12: 2D design of the case

Figure 4.13: 3D design of the case

v Software

1. Keil-C and Pico Template

We will use Keil-C compiler, along with µVision IDE. Since there is ARM Cortex M0+ support, but no direct support for Raspberry Pi Pico, we need to rely on a third-party template. GorgonMeducer's Pico_Template [8], which available free on GitHub, provide an "out-of-the-box" experience for newbie developers use Pico with Keil.

Assume you have installed packages for ARM Cortex M0+, to use GorgonMeducer's Pico_Template, clone the repository with recursive mode to clone entire repo (including pico-sdk). Then install packages that end with `.pack` extension.

```
git clone https://github.com/GorgonMeducer/Pico_Template --recursive
```

The `main.c` file is the main program code that we will write our code on.

2. Built-in functions

- `void gpio_init(uint gpio)`: Initialize a GPIO pin.
- `static void gpio_set_dir (uint gpio, bool out)`: Set a single GPIO direction, `out = GPIO_OUT / GPIO_IN`.
- `static void gpio_put (uint gpio, bool value)`: Drive a single GPIO high/low.
- `static bool gpio_get (uint gpio)`: Get state of a single specified GPIO.
- `void sleep_ms(int milisecond)`: Pause and wait current thread for `milisecond` ms.

3. Implement hardware with code

a. Read states of proximity sensor

```
gpio_get(IR);
```

b. Drive four-digit 7-segment display

First, we define an array for characters from 0 to F and an array for storing data of LEDs.

```
unsigned char LED_0F[] = {
// 0     1      2    3     4     5     6    7     8     9
  0xC0, 0xF9, 0xA4, 0xB0, 0x99, 0x92, 0x82, 0xF8, 0x80, 0x90,
// A     b     C     d     E     F     -    off
  0x8C, 0xBF, 0xC6, 0xA1, 0x86, 0x8E, 0xbf, 0xFF
};

unsigned char LED[4] = {17,17,17,17};
```

`LED_OUT` function will send data bit by bit to `DIO` pin.

```
void LED_OUT(unsigned char X) {
  unsigned char i;
  for (i = 8; i >= 1; i--) {
```

```
    if (X & 0x80) {
      gpio_put(DIO, 1);
    } else {
      gpio_put(DIO, 0);
    }
    X <<= 1;
    gpio_put(SCLK, 0);
    gpio_put(SCLK, 1);
  }
}
```

Example: `X = 0xC0` (displaying 0 on the LED)

- `11000000 and 10000000 = 10000000` $\Longrightarrow$ Output `1` to `DIO`
- `10000000 and 10000000 = 10000000` $\Longrightarrow$ Output `1` to `DIO`
- `00000000 and 10000000 = 00000000` $\Longrightarrow$ Output `0` to `DIO`
- `00000000 and 10000000 = 00000000` $\Longrightarrow$ Output `0` to `DIO`
- ...

As a result,  is displayed on the LED.

The above function output only one character. We need to write a function to output all four digits on the LED. The function takes data of `LED[]` array and output to the 7-segment LEDs. The number to be displayed on the screen will be specified as the corresponding hexadecimal form in `LED_0F` and inserted into the 7-segment display as the binary converted from hexadecimal in `LED_0F`.

```
void LED4_Display(void) {
  unsigned char * led_table;
  unsigned char i;

  led_table = LED_0F + LED[0];
  i = * led_table;
  LED_OUT(i);
  LED_OUT(0x08); // 1000
  gpio_put(RCLK, 0);
  gpio_put(RCLK, 1);

  led_table = LED_0F + LED[1];
  i = * led_table;
  LED_OUT(i);
  LED_OUT(0x04); // 0100
  gpio_put(RCLK, 0);
  gpio_put(RCLK, 1);

  led_table = LED_0F + LED[2];
  i = * led_table;
  LED_OUT(i);
  LED_OUT(0x02); //0010
  gpio_put(RCLK, 0);
  gpio_put(RCLK, 1);

  led_table = LED_0F + LED[3];
  i = * led_table;
  LED_OUT(i);
  LED_OUT(0x01); //0001
  gpio_put(RCLK, 0);
  gpio_put(RCLK, 1);
}
```

Before displaying we need to determine the hexadecimal data to be displayed. `led_table = LED_0F + LED[0]` will point to the hexadecimal data that needed to output to the LED. The procedure is

similar to other LEDs.

c. Separate integer into digits

```
void Num2LED(int num) {
  LED[3] = num % 10;
  LED[2] = (num /= 10) % 10;
  LED[1] = (num /= 10) % 10;
  LED[0] = num / 10;
  LED4_Display();
}
```

Example: 1234

- `LED[3] = num % 10 = 1234 % 10 = 4`
- `LED[2] = (num /= 10) % 10 = 123 % 10 = 3`
- `LED[1] = (num /= 10) % 10 = 12 % 10 = 2`
- `LED[0] = num / 10 = 12 / 10 = 1`

4. Using the buzzer

The buzzer will "beep" for a short period of time, so we will set the buzzer on for 100ms then off.

```
gpio_put(BUZZER, 1);
sleep_ms(100);
gpio_put(BUZZER, 0);
```

VI Implementation result

1. Result

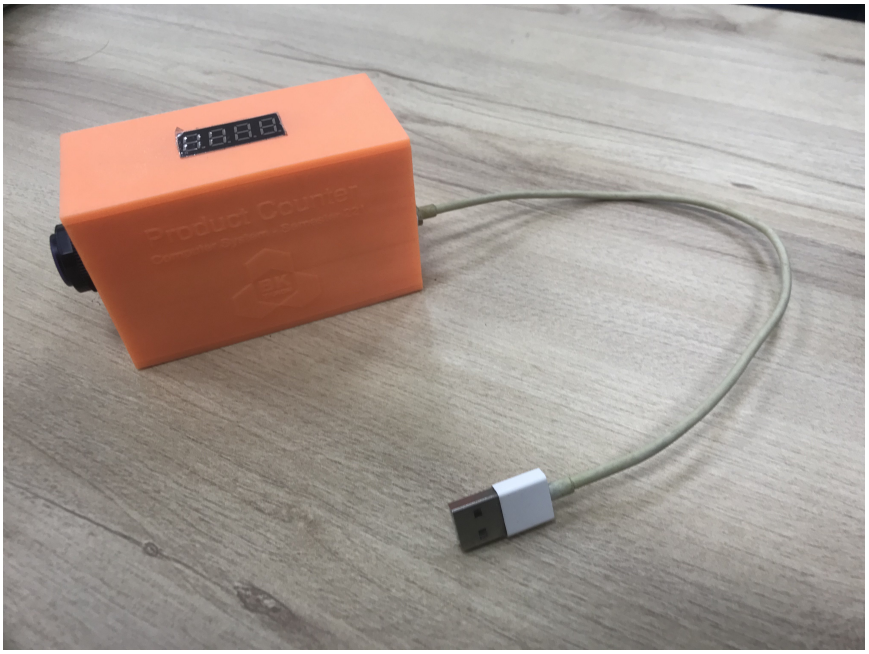


Figure 6.1: The complete device

Figure 6.2: The device counts and "beep" when an object passed by

2. Source code

```
1  // SOME CODES ARE PREDEFINED AND INITIALIZED. DO NOT EDIT.
2  /*============================ INCLUDES ====================================*/
3  #include "pico/stdlib.h"
4  #include "perf_counter.h"
5
6  #if defined(__PICO_USE_LCD_1IN3__) && __PICO_USE_LCD_1IN3__
7  #include "DEV_Config.h"
8  #include "LCD_1In3.h"
9  #include "GLCD_Config.h"
10 #endif
```

```
11
12  #include <stdio.h>
13  #include "RTE_Components.h"
14
15  #if defined(RTE_Compiler_EventRecorder) && defined(USE_EVR_FOR_STDOUR)
16  # include < EventRecorder.h >
17  #endif
18
19  /*============================ MACROS ========================================*/
20  #define TOP        (0x1FFF)
21
22  /*============================ MACROFIED FUNCTIONS ============================*/
23  #define ABS(__N)((__N) < 0 ? -(__N) : (__N))
24  #define _BV(__N)((uint32_t) 1 << (__N))
25  #define IR 16
26  #define BUZZER 17
27  #define DIO 18
28  #define RCLK 19
29  #define SCLK 20
30
31  /*============================ TYPES ==========================================*/
32  /*============================ GLOBAL VARIABLES ===============================*/
33  unsigned char LED_0F[] = {
34  // 0     1      2    3     4     5      6    7     8     9
35    0xC0, 0xF9, 0xA4, 0xB0, 0x99, 0x92, 0x82, 0xF8, 0x80, 0x90,
36  // A     b     C     d     E     F     -    off
37    0x8C, 0xBF, 0xC6, 0xA1, 0x86, 0x8E, 0xbf, 0xFF
38  };
39
40  unsigned char LED[4] = {};
41  /*============================ LOCAL VARIABLES ================================*/
42  /*============================ PROTOTYPES =====================================*/
43  /*============================ IMPLEMENTATION =================================*/
44
45  void SysTick_Handler(void) {}
46
47  static void system_init(void) {
48    extern void SystemCoreClockUpdate();
49
50    SystemCoreClockUpdate();
51    /*! \note if you do want to use SysTick in your application, please use
52     *!       init_cycle_counter(true);
53     *!       instead of
54     *!       init_cycle_counter(false);
55     */
56    init_cycle_counter(false);
57
58    #if defined(RTE_Compiler_EventRecorder) && defined(USE_EVR_FOR_STDOUR)
59    EventRecorderInitialize(0, 1);
60    #endif
61
62    gpio_init(DIO);
63    gpio_set_dir(DIO, GPIO_OUT);
64
65    gpio_init(SCLK);
66    gpio_set_dir(SCLK, GPIO_OUT);
67
68    gpio_init(RCLK);
69    gpio_set_dir(RCLK, GPIO_OUT);
70
71    gpio_init(BUZZER);
72    gpio_set_dir(BUZZER, GPIO_OUT);
73
74    gpio_init(IR);
75    gpio_set_dir(IR, GPIO_IN);
76
77    #if defined(__PICO_USE_LCD_1IN3__) && __PICO_USE_LCD_1IN3__
78    DEV_Delay_ms(100);
79
```

```
  if (DEV_Module_Init() != 0) {
    //assert(0);
  }

  DEV_SET_PWM(50);
  /* LCD Init */

  LCD_1IN3_Init(HORIZONTAL);
  LCD_1IN3_Clear(GLCD_COLOR_BLUE);

  for (int n = 0; n < KEY_NUM; n++) {
    dev_key_init(n);
  }
  #endif
}

void LED_OUT(unsigned char X) {
  unsigned char i;
  for (i = 8; i >= 1; i--) {
    if (X & 0x80) {
      gpio_put(DIO, 1);
    } else {
      gpio_put(DIO, 0);
    }
    X <<= 1;
    gpio_put(SCLK, 0);
    gpio_put(SCLK, 1);
  }
}

void LED4_Display(void) {
  unsigned char * led_table;
  unsigned char i;

  led_table = LED_0F + LED[0];
  i = * led_table;
  LED_OUT(i);
  LED_OUT(0x08); // 1000
  gpio_put(RCLK, 0);
  gpio_put(RCLK, 1); // store in shift register

  led_table = LED_0F + LED[1];
  i = * led_table;
  LED_OUT(i);
  LED_OUT(0x04); // 0100
  gpio_put(RCLK, 0);
  gpio_put(RCLK, 1);

  led_table = LED_0F + LED[2];
  i = * led_table;
  LED_OUT(i);
  LED_OUT(0x02); //0010
  gpio_put(RCLK, 0);
  gpio_put(RCLK, 1);

  led_table = LED_0F + LED[3];
  i = * led_table;
  LED_OUT(i);
  LED_OUT(0x01); //0001
  gpio_put(RCLK, 0);
  gpio_put(RCLK, 1);
}

void Num2LED(int num) {
  LED[3] = num % 10;
  LED[2] = (num /= 10) % 10;
  LED[1] = (num /= 10) % 10;
  LED[0] = num / 10;
  LED4_Display();
```

```
}

int main(void) {
  system_init();
  gpio_put(BUZZER, 0);
  sleep_ms(100);
  int count = 0;

  while (true) {
    Num2LED(count);
    if (gpio_get(IR) == 0) {
      count++;
      gpio_put(BUZZER, 1);
      sleep_ms(100);
      gpio_put(BUZZER, 0);

      while (gpio_get(IR) == 0)
        Num2LED(count);
    }
  }

  return 0;
}
```

VII Conclusion

We have successfully created the product counter by using Raspberry Pi Pico. We also discover the working principle of the infrared proximity sensor, four-digit 7-segment display, and the buzzer. Furthermore, we have designed the 3D model case by using Autodesk Fusion 360. But the power supply devices still depend on the external source (5V phone charger, power bank, computer,...). In the future, we will integrate the battery into the device, create a button to reset the counter, optimize the size of the 3D printed case, and increase the accuracy of the sensor.

VIII References

[1] *CMOS Shift Register TM74HC59*. Titan Micro Electronics.

[2] Digilent. *Learn.Digilentinc — Music with a Piezoelectric Buzzer*. URL: https://learn.digilentinc.com/Documents/392.

[3] Electronics Hub. *IR (Infrared) Obstacle Detection Sensor Circuit*. 2015. URL: https://www.electronicshub.org/ir-sensor/.

[4] Nikhil Joshi. *What are multiplexed displays and use of multiplexing - Arduino - C Programming,Seven Segment - Dotnetlovers*. 2020. URL: https://www.dotnetlovers.com/article/10246/what-are-multiplexed-displays-and-use-of-multiplexing.

[5] Raspberry Pi Ltd. *Pico-R3-A4-Pinout*. 2022. URL: https://datasheets.raspberrypi.com/pico/Pico-R3-A4-Pinout.pdf.

[6] *Raspberry Pi Pico C/C++ SDK*. 1.4.0. Raspberry Pi Ltd. 06/2022.

[7] *Raspberry Pi Pico Datasheet*. 1.9. Raspberry Pi Ltd. 06/2022.

[8] Gabriel Wang. *Pico Template*. 2022. URL: https://github.com/GorgonMeducer/Pico_Template.

[9] Waveshare. *Raspberry Pi Pico - Waveshare Wiki*. 2022. URL: https://www.waveshare.com/wiki/Raspberry_Pi_Pico.

[10] Wikipedia. *ARM architecture family - Wikipedia*. 2022. URL: http://en.wikipedia.org/wiki/ARM_architecture_family.

[11] Wikipedia. *Buzzer - Wikipedia*. 2022. URL: https://en.wikipedia.org/wiki/Buzzer.

[12] Wikipedia. *Microcontroller - Wikipedia*. 2022. URL: http://en.wikipedia.org/wiki/Microcontroller.

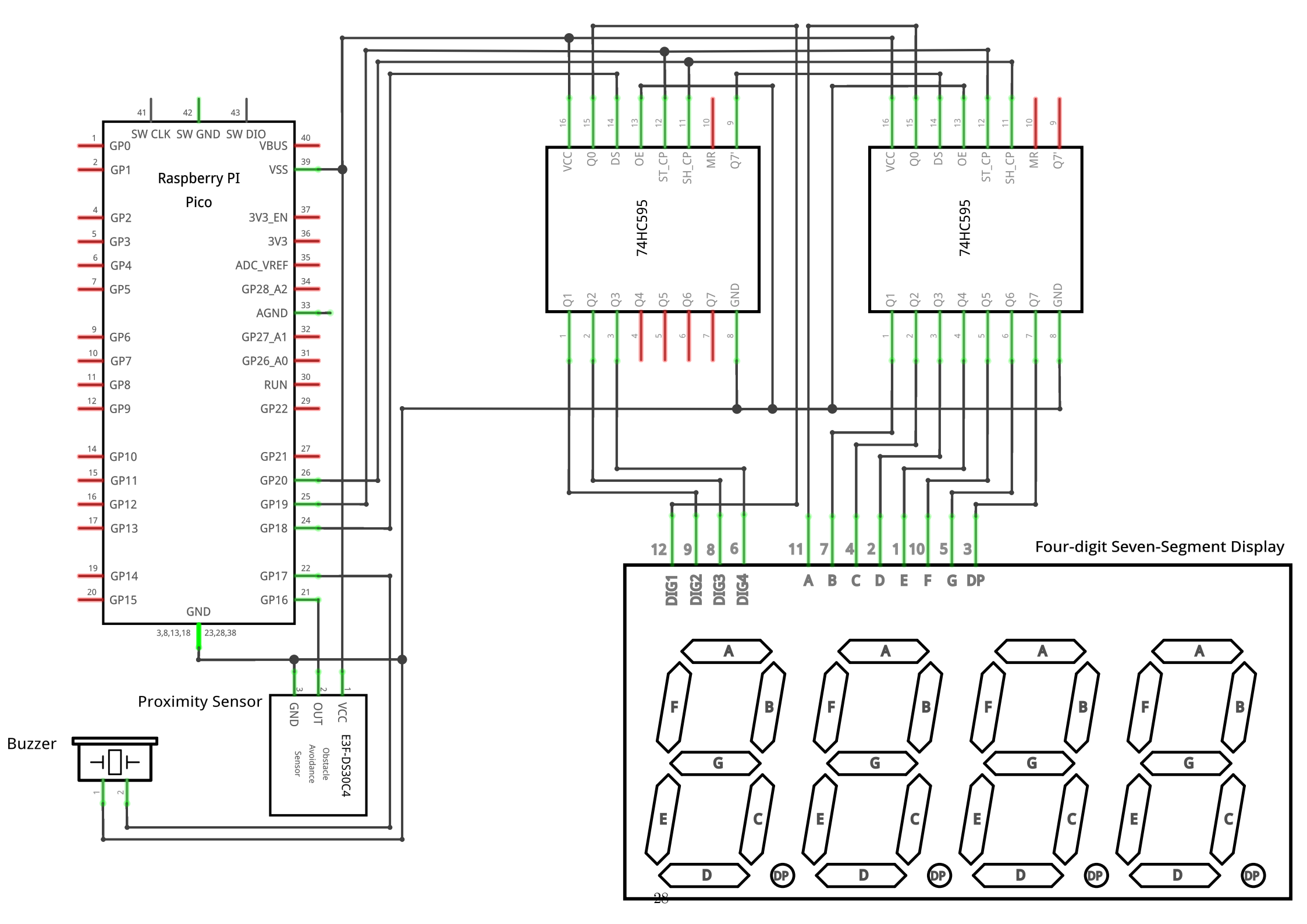
[13] Wikipedia. *Multiplexed display - Wikipedia*. 2022. URL: https://en.wikipedia.org/wiki/Multiplexed_display.

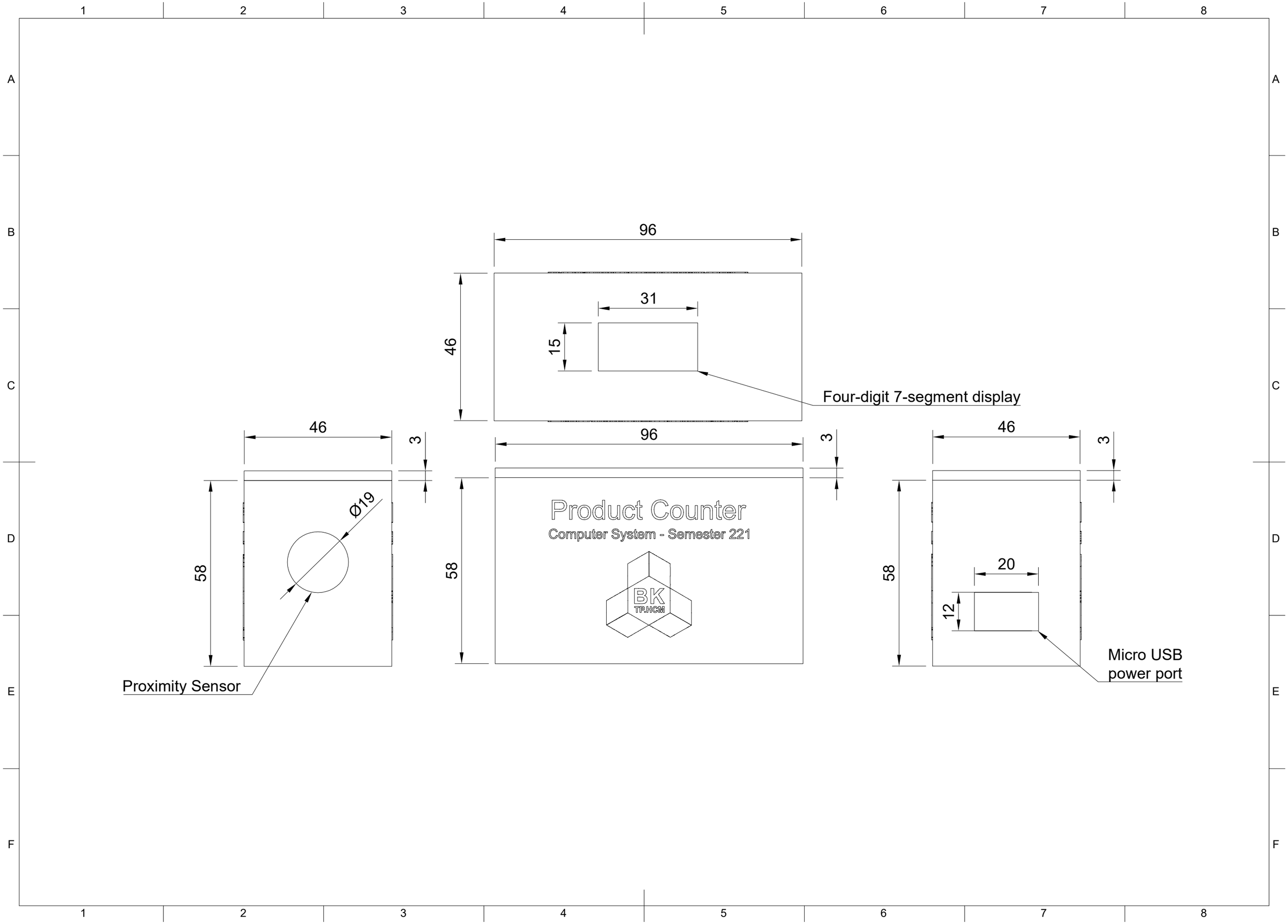
[14] Wikipedia. *Seven-segment display - Wikipedia*. 2022. URL: https://en.wikipedia.org/wiki/Seven-segment_display.

IX Appendix

A larger version of wiring schematic and CAD design of 3D Printed case (1:1 scale) are listed below.

All the program source codes and reports' LaTeX source codes can be accessed at our GitHub repository: `https://github.com/superzeldalink/ProductCounter-Pico`.


Raspberry PI
Pico
SW CLK
SW GND
SW DIO
GP0
GP1
GP2
GP3
GP4
GP5
GP6
GP7
GP8
GP9
GP10
GP11
GP12
GP13
GP14
GP15
GND
VBUS
VSS
3V3_EN
3V3
ADC_VREF
GP28_A2
AGND
GP27_A1
GP26_A0
RUN
GP22
GP21
GP20
GP19
GP18
GP17
GP16
74HC595
VCC
Q0
DS
OE
ST_CP
SH_CP
MR
Q7'
Q1
Q2
Q3
Q4
Q5
Q6
Q7
GND
Four-digit Seven-Segment Display
DIG1
DIG2
DIG3
DIG4
A
B
C
D
E
F
G
DP
Proximity Sensor
E3F-DS30C4
Obstacle Avoidance Sensor
OUT
Buzzer


96
31
46
15
Four-digit 7-segment display
46
3
96
3
46
3
Ø19
58
58
58
Product Counter
Computer System - Semester 221
BK
TP.HCM
20
12
Proximity Sensor
Micro USB
power port