CPE316 – Embedded Systems Final Project Report

DTCL-Distance and Time Controlled Lock

Semester II (2021-2022)

Course supervisors:	Team LAMBDA members:
Dr. Iman Elawady ENG. Michael B.Khani	*Muhammet Özer (2017010206037@ogrenci.karabuk.edu.tr) Adem Onur Çelik (1910205022@ogrenci.karabuk.edu.tr) Hatice Gül Çoban (1810205036)(haticegcobann@hotmail.com) Feride Nur Yaman (1810205069@ogrenci.karabuk.edu.tr) Ayfer Gözel (2017010216013@ogrenci.karabuk.edu.tr) Bayram Enes Yıldırım (2017010206020@ogrenci.karabuk.edu.tr) İrem Yargı (1910206527@ogrenci.karabuk.edu.tr) Ahmet Köşker (2017010206022@ogrenci.karabuk.edu.tr)

* Team leader and correspondent member

Contents

1. Introduction

Main purpose of DTCL is to provide home security by controlling locks. These locks can be used with windows, drawers, doors etc.

When activated if someone is at home or time is before midnight all locks are unlocked. If someone isn't home or time is after midnight all locks are locked. But what if we want to use them after midnight? Distance sensor will be remained active and let you unlock if you're closer than 40 centimeters.

The project consists of 3 main parts. ESP32, servo motor and ultrasonic distance sensor. An LCD display can be used to show time, but this is optional.

2. Related Works

We can choose which pins on the ESP32 are UART, I2C, or SPI by setting them in the code. We've used I2C LCD, HCSR04 distance sensor, and servo motor.

This is feasible because to the multiplexing capability of the ESP32 chip, which allows many functions to be assigned to the single pin.

3. Project Design

We've discussed about the steps to complete. Second we designed the circuit then programmed the esp32 with necessary sensors. Finally we've done many tests to ensure that project works.

3.1. Project Layout

We've prepared a presentation instead of the project infographic.

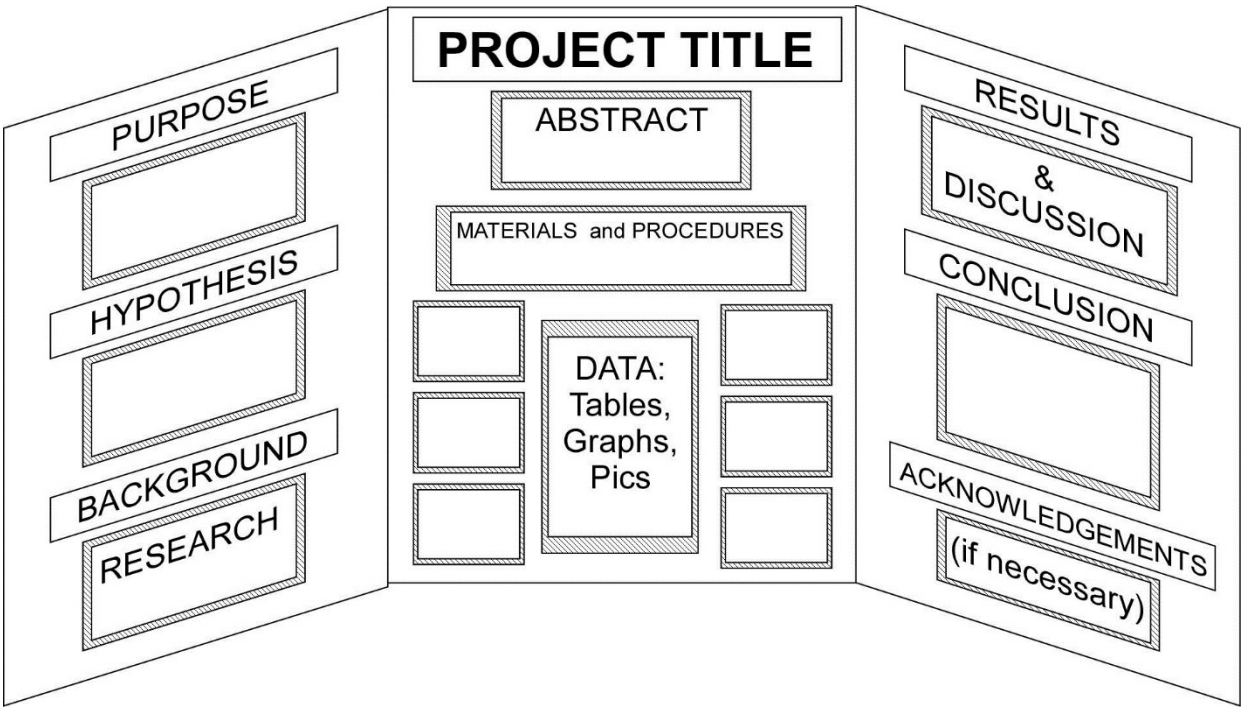
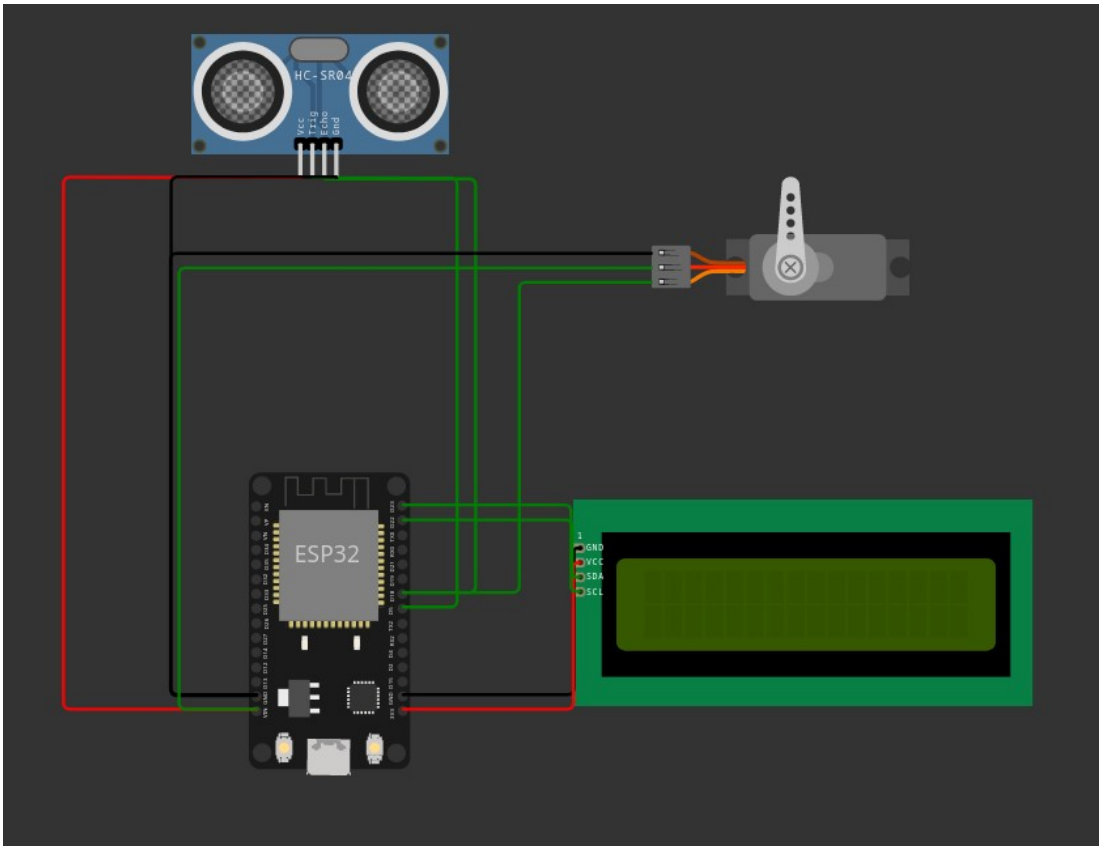


Figure 1 – Just a sample to see how to demonstrate an infographic

3.2. Project Mechanism

"A mechanism for a behavior is a complex system that produces that behavior by the interaction of a number of parts, where the interactions between parts can be characterized by direct, invariant, change-relating generalizations", As Prof. Glennan said [1].



Here, the project mechanism, diagrams and connections will be demonstrated using suitable figures and descriptions.

4. Conclusion and Results

For the first time in a project, we had used micropython. We spoke about the issues and figured out how to solve them. We tried to work together.

5. Lesson Learnt

We had experienced micropyhton for the first time in a project. We discussed about the problems and we learned how to overcome them. We tried to be team.

6. References

[1] Illari, P. M., & Williamson, J., "What is a mechanism? Thinking about mechanisms across the sciences.," *European Journal for Philosophy of Science*, , vol. 2, no. 1, pp. 119-135, 2012.

6.1. Code Modules

6.1.1. main.py

```
 1  import machine
 2  from machine import Pin, SoftI2C,PWM
 3  from lcd_api import LcdApi
 4  from i2c_lcd import I2cLcd
 5  from time import sleep
 6  from hcsr04 import HCSR04
 7
 8  # I2C LCD sensor and parameters
 9  I2C_ADDR = 0x27
10  totalRows = 2
11  totalColumns = 16
12
13  i2c = SoftI2C(scl=Pin(23), sda=Pin(22), freq=5000)
14  #initializing the I2C method for ESP32
15  lcd = I2cLcd(i2c, I2C_ADDR, totalRows, totalColumns)
16
17  lcd.putstr("WELCOME")
18  sleep(2)
19  lcd.clear()
20  lcd.putstr("Project LAMBDA")
21  sleep(2)
22  lcd.clear()
23
24  # HCSR04 Sensor and parameters
25  sensor = HCSR04(trigger_pin=5, echo_pin=18,
26  echo_timeout_us=10000)
27
28  servo = PWM (Pin(18), freq=50)
29
30  while True:
31    distance = sensor.distance_cm()
32    sleep(1)
33    servo.duty(1)
34    lcd.clear()
35    if (distance<40):
36      lcd.putstr("Lock enabled.")
37      sleep(2)
38      lcd.clear()
39      servo.duty(90)
40    else:
```

```
41      lcd.putstr("Lock disabled.")
42      sleep(2)
43      servo.duty(1)
```

6.1.2. hcsr04.py

```
1   import machine, time
2   from machine import Pin
3
4   __version__ = '0.2.0'
5   __author__ = 'Roberto Sánchez'
6   __license__ = "Apache License 2.0.
7   https://www.apache.org/licenses/LICENSE-2.0"
8
9   class HCSR04:
10      """
11      Driver to use the untrasonic sensor HC-SR04.
12      The sensor range is between 2cm and 4m.
13      The timeouts received listening to echo pin are
14  converted to OSError('Out of range')
15      """
16      # echo_timeout_us is based in chip range limit (400cm)
17      def __init__(self, trigger_pin, echo_pin,
18  echo_timeout_us=500*2*30):
19          """
20          trigger_pin: Output pin to send pulses
21          echo_pin: Readonly pin to measure the distance. The
22  pin should be protected with 1k resistor
23          echo_timeout_us: Timeout in microseconds to listen
24  to echo pin.
25          By default is based in sensor limit range (4m)
26          """
27          self.echo_timeout_us = echo_timeout_us
28          # Init trigger pin (out)
29          self.trigger = Pin(trigger_pin, mode=Pin.OUT,
30  pull=None)
31          self.trigger.value(0)
32
33          # Init echo pin (in)
34          self.echo = Pin(echo_pin, mode=Pin.IN, pull=None)
35
```

```
    def _send_pulse_and_wait(self):
        """
        Send the pulse to trigger and listen on echo pin.
        We use the method `machine.time_pulse_us()` to get
the microseconds until the echo is received.
        """
        self.trigger.value(0) # Stabilize the sensor
        time.sleep_us(5)
        self.trigger.value(1)
        # Send a 10us pulse.
        time.sleep_us(10)
        self.trigger.value(0)
        try:
            pulse_time = machine.time_pulse_us(self.echo, 1,
self.echo_timeout_us)
            return pulse_time
        except OSError as ex:
            if ex.args[0] == 110: # 110 = ETIMEDOUT
                raise OSError('Out of range')
            raise ex

    def distance_mm(self):
        """
        Get the distance in milimeters without floating
point operations.
        """
        pulse_time = self._send_pulse_and_wait()

        # To calculate the distance we get the pulse_time
and divide it by 2
        # (the pulse walk the distance twice) and by 29.1
becasue
        # the sound speed on air (343.2 m/s), that It's
equivalent to
        # 0.34320 mm/us that is 1mm each 2.91us
        # pulse_time // 2 // 2.91 -> pulse_time // 5.82 ->
pulse_time * 100 // 582
        mm = pulse_time * 100 // 582
        return mm

    def distance_cm(self):
        """
        Get the distance in centimeters with floating point
```

```
79  operations.
80          It returns a float
81          """
82          pulse_time = self._send_pulse_and_wait()
83
84          # To calculate the distance we get the pulse_time
85  and divide it by 2
86          # (the pulse walk the distance twice) and by 29.1
87  becasue
88          # the sound speed on air (343.2 m/s), that It's
89  equivalent to
90          # 0.034320 cm/us that is 1cm each 29.1us
91          cms = (pulse_time / 2) / 29.1
92          return cms
```

6.1.3. i2c_lcd.py

```
1   import utime
2   import gc
3
4   from lcd_api import LcdApi
5   from machine import I2C
6
7   # PCF8574 pin definitions
8   MASK_RS = 0x01       # P0
9   MASK_RW = 0x02       # P1
10  MASK_E  = 0x04       # P2
11
12  SHIFT_BACKLIGHT = 3  # P3
13  SHIFT_DATA      = 4  # P4-P7
14
15  class I2cLcd(LcdApi):
16
17      #Implements a HD44780 character LCD connected via
18  PCF8574 on I2C
19
20      def __init__(self, i2c, i2c_addr, num_lines,
21  num_columns):
22          self.i2c = i2c
23          self.i2c_addr = i2c_addr
24          self.i2c.writeto(self.i2c_addr, bytes([0]))
25          utime.sleep_ms(20)   # Allow LCD time to powerup
26          # Send reset 3 times
```

```
        self.hal_write_init_nibble(self.LCD_FUNCTION_RESET)
        utime.sleep_ms(5)    # Need to delay at least 4.1
msec
        self.hal_write_init_nibble(self.LCD_FUNCTION_RESET)
        utime.sleep_ms(1)
        self.hal_write_init_nibble(self.LCD_FUNCTION_RESET)
        utime.sleep_ms(1)
        # Put LCD into 4-bit mode
        self.hal_write_init_nibble(self.LCD_FUNCTION)
        utime.sleep_ms(1)
        LcdApi.__init__(self, num_lines, num_columns)
        cmd = self.LCD_FUNCTION
        if num_lines > 1:
            cmd |= self.LCD_FUNCTION_2LINES
        self.hal_write_command(cmd)
        gc.collect()

    def hal_write_init_nibble(self, nibble):
        # Writes an initialization nibble to the LCD.
        # This particular function is only used during
initialization.
        byte = ((nibble >> 4) & 0x0f) << SHIFT_DATA
        self.i2c.writeto(self.i2c_addr, bytes([byte |
MASK_E]))
        self.i2c.writeto(self.i2c_addr, bytes([byte]))
        gc.collect()

    def hal_backlight_on(self):
        # Allows the hal layer to turn the backlight on
        self.i2c.writeto(self.i2c_addr, bytes([1 <<
SHIFT_BACKLIGHT]))
        gc.collect()

    def hal_backlight_off(self):
        #Allows the hal layer to turn the backlight off
        self.i2c.writeto(self.i2c_addr, bytes([0]))
        gc.collect()

    def hal_write_command(self, cmd):
        # Write a command to the LCD. Data is latched on the
falling edge of E.
        byte = ((self.backlight << SHIFT_BACKLIGHT) |
                (((cmd >> 4) & 0x0f) << SHIFT_DATA))
```

```
70          self.i2c.writeto(self.i2c_addr, bytes([byte |
71  MASK_E]))
72          self.i2c.writeto(self.i2c_addr, bytes([byte]))
73          byte = ((self.backlight << SHIFT_BACKLIGHT) |
74                  ((cmd & 0x0f) << SHIFT_DATA))
75          self.i2c.writeto(self.i2c_addr, bytes([byte |
76  MASK_E]))
77          self.i2c.writeto(self.i2c_addr, bytes([byte]))
78          if cmd <= 3:
79              # The home and clear commands require a worst
80  case delay of 4.1 msec
81              utime.sleep_ms(5)
82          gc.collect()
83
84      def hal_write_data(self, data):
85          # Write data to the LCD. Data is latched on the
86  falling edge of E.
87          byte = (MASK_RS |
88                  (self.backlight << SHIFT_BACKLIGHT) |
89                  (((data >> 4) & 0x0f) << SHIFT_DATA))
90          self.i2c.writeto(self.i2c_addr, bytes([byte |
91  MASK_E]))
92          self.i2c.writeto(self.i2c_addr, bytes([byte]))
93          byte = (MASK_RS |
94                  (self.backlight << SHIFT_BACKLIGHT) |
95                  ((data & 0x0f) << SHIFT_DATA))
96          self.i2c.writeto(self.i2c_addr, bytes([byte |
97  MASK_E]))
98          self.i2c.writeto(self.i2c_addr, bytes([byte]))
99          gc.collect()
```

6.1.4. lcdapi.py

```
1   import time
2
3   class LcdApi:
4       """Implements the API for talking with HD44780
5   compatible character LCDs.
6       This class only knows what commands to send to the LCD,
7   and not how to get
8       them to the LCD.
9
10      It is expected that a derived class will implement the
```

```
11  hal_xxx functions.
12      """
13
14      # The following constant names were lifted from the
15  avrlib lcd.h
16      # header file, however, I changed the definitions from
17  bit numbers
18      # to bit masks.
19      #
20      # HD44780 LCD controller command set
21
22      LCD_CLR = 0x01              # DB0: clear display
23      LCD_HOME = 0x02             # DB1: return to home
24  position
25
26      LCD_ENTRY_MODE = 0x04       # DB2: set entry mode
27      LCD_ENTRY_INC = 0x02        # --DB1: increment
28      LCD_ENTRY_SHIFT = 0x01      # --DB0: shift
29
30      LCD_ON_CTRL = 0x08          # DB3: turn lcd/cursor on
31      LCD_ON_DISPLAY = 0x04       # --DB2: turn display on
32      LCD_ON_CURSOR = 0x02        # --DB1: turn cursor on
33      LCD_ON_BLINK = 0x01         # --DB0: blinking cursor
34
35      LCD_MOVE = 0x10             # DB4: move cursor/display
36      LCD_MOVE_DISP = 0x08        # --DB3: move display (0->
37  move cursor)
38      LCD_MOVE_RIGHT = 0x04       # --DB2: move right (0->
39  left)
40
41      LCD_FUNCTION = 0x20         # DB5: function set
42      LCD_FUNCTION_8BIT = 0x10    # --DB4: set 8BIT mode (0-
43  >4BIT mode)
44      LCD_FUNCTION_2LINES = 0x08  # --DB3: two lines (0->one
45  line)
46      LCD_FUNCTION_10DOTS = 0x04  # --DB2: 5x10 font (0->5x7
47  font)
48      LCD_FUNCTION_RESET = 0x30   # See "Initializing by
49  Instruction" section
50
51      LCD_CGRAM = 0x40            # DB6: set CG RAM address
52      LCD_DDRAM = 0x80            # DB7: set DD RAM address
53
```

```
    LCD_RS_CMD = 0
    LCD_RS_DATA = 1

    LCD_RW_WRITE = 0
    LCD_RW_READ = 1

    def __init__(self, num_lines, num_columns):
        self.num_lines = num_lines
        if self.num_lines > 4:
            self.num_lines = 4
        self.num_columns = num_columns
        if self.num_columns > 40:
            self.num_columns = 40
        self.cursor_x = 0
        self.cursor_y = 0
        self.implied_newline = False
        self.backlight = True
        self.display_off()
        self.backlight_on()
        self.clear()
        self.hal_write_command(self.LCD_ENTRY_MODE |
self.LCD_ENTRY_INC)
        self.hide_cursor()
        self.display_on()

    def clear(self):
        """Clears the LCD display and moves the cursor to
the top left
        corner.
        """
        self.hal_write_command(self.LCD_CLR)
        self.hal_write_command(self.LCD_HOME)
        self.cursor_x = 0
        self.cursor_y = 0

    def show_cursor(self):
        """Causes the cursor to be made visible."""
        self.hal_write_command(self.LCD_ON_CTRL |
self.LCD_ON_DISPLAY |
                               self.LCD_ON_CURSOR)

    def hide_cursor(self):
        """Causes the cursor to be hidden."""
```

```
 97          self.hal_write_command(self.LCD_ON_CTRL |
 98      self.LCD_ON_DISPLAY)
 99
100      def blink_cursor_on(self):
101          """Turns on the cursor, and makes it blink."""
102          self.hal_write_command(self.LCD_ON_CTRL |
103      self.LCD_ON_DISPLAY |
104                                 self.LCD_ON_CURSOR |
105      self.LCD_ON_BLINK)
106
         def blink_cursor_off(self):
             """Turns on the cursor, and makes it no blink (i.e.
         be solid)."""
             self.hal_write_command(self.LCD_ON_CTRL |
         self.LCD_ON_DISPLAY |
                                     self.LCD_ON_CURSOR)

         def display_on(self):
             """Turns on (i.e. unblanks) the LCD."""
             self.hal_write_command(self.LCD_ON_CTRL |
         self.LCD_ON_DISPLAY)

         def display_off(self):
             """Turns off (i.e. blanks) the LCD."""
             self.hal_write_command(self.LCD_ON_CTRL)

         def backlight_on(self):
             """Turns the backlight on.

             This isn't really an LCD command, but some modules
         have backlight
             controls, so this allows the hal to pass through the
         command.
             """
             self.backlight = True
             self.hal_backlight_on()

         def backlight_off(self):
             """Turns the backlight off.

             This isn't really an LCD command, but some modules
         have backlight
             controls, so this allows the hal to pass through the
```

```
command.
        """
        self.backlight = False
        self.hal_backlight_off()

    def move_to(self, cursor_x, cursor_y):
        """Moves the cursor position to the indicated
position. The cursor
        position is zero based (i.e. cursor_x == 0 indicates
first column).
        """
        self.cursor_x = cursor_x
        self.cursor_y = cursor_y
        addr = cursor_x & 0x3f
        if cursor_y & 1:
            addr += 0x40    # Lines 1 & 3 add 0x40
        if cursor_y & 2:    # Lines 2 & 3 add number of
columns
            addr += self.num_columns
        self.hal_write_command(self.LCD_DDRAM | addr)

    def putchar(self, char):
        """Writes the indicated character to the LCD at the
current cursor
        position, and advances the cursor by one position.
        """
        if char == '\n':
            if self.implied_newline:
                # self.implied_newline means we advanced due
to a wraparound,
                # so if we get a newline right after that we
ignore it.
                pass
            else:
                self.cursor_x = self.num_columns
        else:
            self.hal_write_data(ord(char))
            self.cursor_x += 1
        if self.cursor_x >= self.num_columns:
            self.cursor_x = 0
            self.cursor_y += 1
            self.implied_newline = (char != '\n')
        if self.cursor_y >= self.num_lines:
```

```
            self.cursor_y = 0
        self.move_to(self.cursor_x, self.cursor_y)

    def putstr(self, string):
        """Write the indicated string to the LCD at the
current cursor
        position and advances the cursor position
appropriately.
        """
        for char in string:
            self.putchar(char)

    def custom_char(self, location, charmap):
        """Write a character to one of the 8 CGRAM
locations, available
        as chr(0) through chr(7).
        """
        location &= 0x7
        self.hal_write_command(self.LCD_CGRAM | (location <<
3))
        self.hal_sleep_us(40)
        for i in range(8):
            self.hal_write_data(charmap[i])
            self.hal_sleep_us(40)
        self.move_to(self.cursor_x, self.cursor_y)

    def hal_backlight_on(self):
        """Allows the hal layer to turn the backlight on.

        If desired, a derived HAL class will implement this
function.
        """
        pass

    def hal_backlight_off(self):
        """Allows the hal layer to turn the backlight off.

        If desired, a derived HAL class will implement this
function.
        """
        pass

    def hal_write_command(self, cmd):
```

```
        """Write a command to the LCD.

        It is expected that a derived HAL class will
implement this
        function.
        """
        raise NotImplementedError

    def hal_write_data(self, data):
        """Write data to the LCD.

        It is expected that a derived HAL class will
implement this
        function.
        """
        raise NotImplementedError

    def hal_sleep_us(self, usecs):
        """Sleep for some time (given in microseconds)."""
        time.sleep_us(usecs)
```