ESD Essentials Term Paper Evaluation

Generated by Doxygen 1.14.0

Chapter 1

Namespace Index

1.1 Package List

Here are the packages with brief descriptions (if available):

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

Chapter 4

Namespace Documentation

4.1 test1 Namespace Reference

Classes

- class SensorApp

  A GUI application for real-time visualization and monitoring of sensor data.

Functions

- read_ultrasonic ()

  Reads distance (in cm) from the HC-SR04 ultrasonic sensor.
- read_csv_data ()

  Reads data from the CSV log file.

Variables

- int DHT_PIN = 17

  DHT11 data pin.
- int GAS_DO_PIN = 27

  MQ Gas Sensor digital output pin.
- int ULTRASONIC_TRIG = 23

  Ultrasonic sensor TRIG pin.
- int ULTRASONIC_ECHO = 24

  Ultrasonic sensor ECHO pin.
- DHT_SENSOR = Adafruit_DHT.DHT11

  Sensor type for DHT.
- str filename = "essentials_log.csv"

  CSV filename for data logging.
- file_lock = threading.Lock()

  Thread lock to synchronize file access.
- mode
- newline
- writer = csv.writer(file)
- root = tk.Tk()
- app = SensorApp(root)

4.1.1 Function Documentation

4.1.1.1 read_csv_data()

```
test1.read_csv_data ()
```

Reads data from the CSV log file.

Returns

Four lists containing timestamps, temperature, gas, and level data.

Definition at line 117 of file test1.py.

4.1.1.2 read_ultrasonic()

```
test1.read_ultrasonic ()
```

Reads distance (in cm) from the HC-SR04 ultrasonic sensor.

Returns

Distance in centimeters (float), or None if timeout occurs.

Definition at line 88 of file test1.py.

Here is the caller graph for this function:

4.1.2 Variable Documentation

4.1.2.1 app

```
test1.app = SensorApp(root)
```

Definition at line 324 of file test1.py.

4.1.2.2 DHT_PIN

```
int test1.DHT_PIN = 17
```

DHT11 data pin.

Definition at line 43 of file test1.py.

4.1.2.3 DHT_SENSOR

```
test1.DHT_SENSOR = Adafruit_DHT.DHT11
```

Sensor type for DHT.

Definition at line 51 of file test1.py.

4.1.2.4 file_lock

```
test1.file_lock = threading.Lock()
```

Thread lock to synchronize file access.

Definition at line 67 of file test1.py.

4.1.2.5 filename

```
test1.filename = "essentials_log.csv"
```

CSV filename for data logging.

Definition at line 64 of file test1.py.

4.1.2.6 GAS_DO_PIN

```
int test1.GAS_DO_PIN = 27
```

MQ Gas Sensor digital output pin.

Definition at line 45 of file test1.py.

4.1.2.7 mode

```
test1.mode
```

Definition at line 72 of file test1.py.

4.1.2.8 newline

```
test1.newline
```

Definition at line 78 of file test1.py.

4.1.2.9 root

```
test1.root = tk.Tk()
```

Definition at line 323 of file test1.py.

4.1.2.10 ULTRASONIC_ECHO

```
int test1.ULTRASONIC_ECHO = 24
```

Ultrasonic sensor ECHO pin.

Definition at line 49 of file test1.py.

4.1.2.11 ULTRASONIC_TRIG

```
int test1.ULTRASONIC_TRIG = 23
```

Ultrasonic sensor TRIG pin.

Definition at line 47 of file test1.py.

4.1.2.12 writer

```
test1.writer = csv.writer(file)
```

Definition at line 79 of file test1.py.

Chapter 5

Class Documentation

5.1 test1.SensorApp Class Reference

A GUI application for real-time visualization and monitoring of sensor data.

Public Member Functions

- __init__ (self, root)

  Constructor for initializing the GUI layout and logic.
- update_temp_label (self, event=None)

  Updates temperature slider label.
- update_gas_label (self, event=None)

  Updates gas slider label.
- start_sensor_thread (self)

  Starts a background thread for continuous sensor data acquisition.
- sensor_loop (self)

  Sensor data acquisition loop (runs on a background thread).
- update_plots (self)

  Updates live Matplotlib plots on the GUI.
- on_close (self)

  Handles cleanup when closing the GUI.

Public Attributes

- root = root
- temp_threshold = tk.DoubleVar(value=50)
- gas_threshold = tk.IntVar(value=1)
- list times = [ ]
- list temps = [ ]
- list gas_values = [ ]
- list levels = [ ]
- fig
- axs = plt.subplots(3, 1, figsize=(8, 6), sharex=True)
- canvas = FigureCanvasTkAgg(self.fig, master=root)

- temp_slider = ttk.Scale(root, from_=0, to=100, orient='horizontal', variable=self.temp_threshold, command=self.↩ update_temp_label)
- temp_value_label = ttk.Label(root, text=f"{self.temp_threshold.get():.1f}")
- gas_slider = ttk.Scale(root, from_=0, to=1, orient='horizontal', variable=self.gas_threshold, command=self.↩ update_gas_label)
- gas_value_label = ttk.Label(root, text=f"{int(self.gas_threshold.get())}")
- status_label = ttk.Label(root, text="", font=("Arial", 14))
- on_close
- sensor_thread = threading.Thread(target=self.sensor_loop, daemon=True)
- update_plots

5.1.1 Detailed Description

A GUI application for real-time visualization and monitoring of sensor data.

This class creates a Tkinter window embedding live-updating Matplotlib plots for temperature, gas concentration, and water level readings.
It also continuously logs sensor data into a CSV file using a background thread.

Responsibilities:

- Acquire and log sensor data.
- Dynamically plot readings in real-time.
- Provide fault/anomaly detection.
- Allow runtime adjustment of sensor thresholds.

Definition at line 153 of file test1.py.

5.1.2 Constructor & Destructor Documentation

5.1.2.1 __init__()

```
test1.SensorApp.__init__ (
              self,
              root)
```

Constructor for initializing the GUI layout and logic.

Parameters

	root	The main Tkinter window.

- -

Definition at line 156 of file test1.py.

5.1.3 Member Function Documentation

5.1.3.1 on_close()

```
test1.SensorApp.on_close (
              self)
```

Handles cleanup when closing the GUI.

Definition at line 312 of file test1.py.

5.1.3.2 sensor_loop()

```
test1.SensorApp.sensor_loop (
              self)
```

Sensor data acquisition loop (runs on a background thread).

Reads DHT11, MQ gas sensor, and ultrasonic distance data periodically. Logs results to CSV and updates internal memory.

Definition at line 216 of file test1.py.

Here is the call graph for this function:

5.1.3.3 start_sensor_thread()

```
test1.SensorApp.start_sensor_thread (
              self)
```

Starts a background thread for continuous sensor data acquisition.

Definition at line 208 of file test1.py.

5.1.3.4 update_gas_label()

```
test1.SensorApp.update_gas_label (
              self,
              event = None)
```

Updates gas slider label.

Definition at line 204 of file test1.py.

5.1.3.5 update_plots()

```
test1.SensorApp.update_plots (
              self)
```

Updates live Matplotlib plots on the GUI.

Definition at line 262 of file test1.py.

5.1.3.6 update_temp_label()

```
test1.SensorApp.update_temp_label (
              self,
              event = None)
```

Updates temperature slider label.

Definition at line 200 of file test1.py.

5.1.4 Member Data Documentation

5.1.4.1 axs

```
test1.SensorApp.axs = plt.subplots(3, 1, figsize=(8, 6), sharex=True)
```

Definition at line 171 of file test1.py.

5.1.4.2 canvas

```
test1.SensorApp.canvas = FigureCanvasTkAgg(self.fig, master=root)
```

Definition at line 174 of file test1.py.

5.1.4.3 fig

```
test1.SensorApp.fig
```

Definition at line 171 of file test1.py.

5.1.4.4 gas_slider

```
test1.SensorApp.gas_slider = ttk.Scale(root, from_=0, to=1, orient='horizontal', variable=self.↩
gas_threshold, command=self.update_gas_label)
```

Definition at line 185 of file test1.py.

5.1.4.5 gas_threshold

```
test1.SensorApp.gas_threshold = tk.IntVar(value=1)
```

Definition at line 162 of file test1.py.

5.1.4.6 gas_value_label

```
test1.SensorApp.gas_value_label = ttk.Label(root, text=f"{int(self.gas_threshold.get())}")
```

Definition at line 187 of file test1.py.

5.1.4.7 gas_values

```
test1.SensorApp.gas_values = []
```

Definition at line 167 of file test1.py.

5.1.4.8 levels

```
test1.SensorApp.levels = []
```

Definition at line 168 of file test1.py.

5.1.4.9 on_close

```
test1.SensorApp.on_close
```

Definition at line 197 of file test1.py.

5.1.4.10 root

```
test1.SensorApp.root = root
```

Definition at line 157 of file test1.py.

5.1.4.11 sensor_thread

```
test1.SensorApp.sensor_thread = threading.Thread(target=self.sensor_loop, daemon=True)
```

Definition at line 209 of file test1.py.

5.1.4.12 status_label

```
test1.SensorApp.status_label = ttk.Label(root, text="", font=("Arial", 14))
```

Definition at line 190 of file test1.py.

5.1.4.13 temp_slider

```
test1.SensorApp.temp_slider = ttk.Scale(root, from_=0, to=100, orient='horizontal', variable=self.↩
temp_threshold, command=self.update_temp_label)
```

Definition at line 179 of file test1.py.

5.1.4.14 temp_threshold

```
test1.SensorApp.temp_threshold = tk.DoubleVar(value=50)
```

Definition at line 161 of file test1.py.

5.1.4.15 temp_value_label

```
test1.SensorApp.temp_value_label = ttk.Label(root, text=f"{self.temp_threshold.get():.1f}")
```

Definition at line 181 of file test1.py.

5.1.4.16 temps

```
test1.SensorApp.temps = []
```

Definition at line 166 of file test1.py.

5.1.4.17 times

```
test1.SensorApp.times = []
```

Definition at line 165 of file test1.py.

5.1.4.18 update_plots

```
test1.SensorApp.update_plots
```

Definition at line 309 of file test1.py.

The documentation for this class was generated from the following file:

- test1.py

Chapter 6

File Documentation

6.1 test1.py File Reference

Classes

- class test1.SensorApp

  A GUI application for real-time visualization and monitoring of sensor data.

Namespaces

- namespace test1

Functions

- test1.read_ultrasonic ()

  Reads distance (in cm) from the HC-SR04 ultrasonic sensor.
- test1.read_csv_data ()

  Reads data from the CSV log file.

Variables

- int test1.DHT_PIN = 17

  DHT11 data pin.
- int test1.GAS_DO_PIN = 27

  MQ Gas Sensor digital output pin.
- int test1.ULTRASONIC_TRIG = 23

  Ultrasonic sensor TRIG pin.
- int test1.ULTRASONIC_ECHO = 24

  Ultrasonic sensor ECHO pin.
- test1.DHT_SENSOR = Adafruit_DHT.DHT11

  Sensor type for DHT.
- str test1.filename = "essentials_log.csv"

  CSV filename for data logging.
- test1.file_lock = threading.Lock()

  Thread lock to synchronize file access.
- test1.mode
- test1.newline
- test1.writer = csv.writer(file)
- test1.root = tk.Tk()
- test1.app = SensorApp(root)

6.2 test1.py

Go to the documentation of this file.

```
00001
00025
00026 import tkinter as tk
00027 from tkinter import ttk
00028 from tkinter import messagebox
00029 import RPi.GPIO as GPIO
00030 import Adafruit_DHT
00031 import time
00032 import csv
00033 from datetime import datetime
00034 from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
00035 import matplotlib.pyplot as plt
00036 import threading
00037
00038 # ================================================================
00039 #                   GPIO AND SENSOR CONFIGURATION
00040 # ================================================================
00041
00042
00043 DHT_PIN = 17
00044
00045 GAS_DO_PIN = 27
00046
00047 ULTRASONIC_TRIG = 23
00048
00049 ULTRASONIC_ECHO = 24
00050
00051 DHT_SENSOR = Adafruit_DHT.DHT11
00052
00053 # GPIO mode and pin setup
00054 GPIO.setmode(GPIO.BCM)
00055 GPIO.setup(GAS_DO_PIN, GPIO.IN)
00056 GPIO.setup(ULTRASONIC_TRIG, GPIO.OUT)
00057 GPIO.setup(ULTRASONIC_ECHO, GPIO.IN)
00058
00059 # ================================================================
00060 #                   CSV FILE INITIALIZATION
00061 # ================================================================
00062
00063
00064 filename = "essentials_log.csv"
00065
00066
00067 file_lock = threading.Lock()
00068
00069 # @brief Initialize CSV file with header if empty
00070 with file_lock:
00071     try:
00072         with open(filename, mode='r') as f:
00073             if f.read(1):
00074                 pass
00075             else:
00076                 raise FileNotFoundError
00077     except (FileNotFoundError, IOError):
00078         with open(filename, mode='w', newline='') as file:
00079             writer = csv.writer(file)
00080             writer.writerow(["Timestamp", "TEMP", "PPM", "LEVEL", "Anomaly"])
00081
00082 # ================================================================
00083 #                   SENSOR READING FUNCTIONS
00084 # ================================================================
00085
00086
00088 def read_ultrasonic():
00089     GPIO.output(ULTRASONIC_TRIG, False)
00090     time.sleep(0.1)
00091
00092     GPIO.output(ULTRASONIC_TRIG, True)
00093     time.sleep(0.00001)
00094     GPIO.output(ULTRASONIC_TRIG, False)
00095
00096     pulse_start = time.time()
00097     timeout = pulse_start + 0.04
00098     while GPIO.input(ULTRASONIC_ECHO) == 0:
00099         pulse_start = time.time()
00100         if pulse_start > timeout:
00101             return None
00102
00103     pulse_end = time.time()
00104     timeout = pulse_end + 0.04
00105     while GPIO.input(ULTRASONIC_ECHO) == 1:
00106         pulse_end = time.time()
```

```
00107         if pulse_end > timeout:
00108             return None
00109 
00110     pulse_duration = pulse_end - pulse_start
00111     distance_cm = pulse_duration * 17150
00112     return round(distance_cm, 2)
00113 
00114 
00115 
00117 def read_csv_data():
00118     times, temps, gas_values, levels = [], [], [], []
00119     with file_lock:
00120         try:
00121             with open(filename, mode='r') as file:
00122                 reader = csv.DictReader(file)
00123                 for row in reader:
00124                     try:
00125                         times.append(datetime.strptime(row["Timestamp"], "%Y-%m-%d %H:%M:%S"))
00126                         temps.append(float(row["TEMP"]) if row["TEMP"] != "N/A" else None)
00127                         gas_values.append(int(row["PPM"]))
00128                         levels.append(float(row["LEVEL"]) if row["LEVEL"] != "N/A" else None)
00129                     except Exception:
00130                         continue
00131         except FileNotFoundError:
00132             pass
00133     return times, temps, gas_values, levels
00134 
00135 
00136 # ============================================================
00137 #                   MAIN APPLICATION CLASS
00138 # ============================================================
00139 
00140 
00152 
00153 class SensorApp:
00154 
00156     def __init__(self, root):
00157         self.root = root
00158         self.root.title("Sensor Monitoring and Visualization")
00159 
00160         # Threshold variables
00161         self.temp_threshold = tk.DoubleVar(value=50)
00162         self.gas_threshold = tk.IntVar(value=1)
00163 
00164         # Data containers for live updates
00165         self.times = []
00166         self.temps = []
00167         self.gas_values = []
00168         self.levels = []
00169 
00170         # ---------------- GUI Configuration ----------------
00171         self.fig, self.axs = plt.subplots(3, 1, figsize=(8, 6), sharex=True)
00172         self.fig.tight_layout(pad=3)
00173 
00174         self.canvas = FigureCanvasTkAgg(self.fig, master=root)
00175         self.canvas.get_tk_widget().grid(row=0, column=0, columnspan=6, padx=10, pady=10)
00176 
00177         # Sliders and labels for thresholds
00178         ttk.Label(root, text="Temperature Threshold (°C)").grid(row=1, column=0, sticky="w", padx=10)
00179         self.temp_slider = ttk.Scale(root, from_=0, to=100, orient='horizontal',
      variable=self.temp_threshold, command=self.update_temp_label)
00180         self.temp_slider.grid(row=1, column=1, sticky="ew", padx=(0,5))
00181         self.temp_value_label = ttk.Label(root, text=f"{self.temp_threshold.get():.1f}")
00182         self.temp_value_label.grid(row=1, column=2, sticky="w")
00183 
00184         ttk.Label(root, text="Gas Threshold (PPM, 0 or 1)").grid(row=1, column=3, sticky="w", padx=10)
00185         self.gas_slider = ttk.Scale(root, from_=0, to=1, orient='horizontal',
      variable=self.gas_threshold, command=self.update_gas_label)
00186         self.gas_slider.grid(row=1, column=4, sticky="ew", padx=(0,5))
00187         self.gas_value_label = ttk.Label(root, text=f"{int(self.gas_threshold.get())}")
00188         self.gas_value_label.grid(row=1, column=5, sticky="w")
00189 
00190         self.status_label = ttk.Label(root, text="", font=("Arial", 14))
00191         self.status_label.grid(row=2, column=0, columnspan=6, pady=10)
00192 
00193         # Start live updates
00194         self.update_plots()
00195         self.start_sensor_thread()
00196 
00197         root.protocol("WM_DELETE_WINDOW", self.on_close)
00198 
00199 
00200     def update_temp_label(self, event=None):
00201         self.temp_value_label.config(text=f"{self.temp_threshold.get():.1f}")
00202 
00203 
00204     def update_gas_label(self, event=None):
```

```
00205         self.gas_value_label.config(text=f"{int(round(self.gas_threshold.get()))}")
00206
00207
00208     def start_sensor_thread(self):
00209         self.sensor_thread = threading.Thread(target=self.sensor_loop, daemon=True)
00210         self.sensor_thread.start()
00211
00212
00216     def sensor_loop(self):
00217         while True:
00218             timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
00219
00220             humidity, temperature = Adafruit_DHT.read(DHT_SENSOR, DHT_PIN)
00221             temp_val = temperature if temperature is not None else "N/A"
00222
00223             gas_state = GPIO.input(GAS_DO_PIN)
00224             gas_detected = (gas_state == 0)
00225             ppm_val = 1 if gas_detected else 0
00226
00227             level = read_ultrasonic()
00228             level_val = level if level is not None else "N/A"
00229
00230             # Simple anomaly detection logic
00231             anomaly = "No"
00232             if temp_val == "N/A" or level_val == "N/A":
00233                 anomaly = "Yes"
00234             else:
00235                 if not (0 <= temp_val <= 50) or not (0 <= level_val <= 400):
00236                     anomaly = "Yes"
00237
00238             # Thread-safe CSV writing
00239             with file_lock:
00240                 with open(filename, mode='a', newline='') as file:
00241                     writer = csv.writer(file)
00242                     writer.writerow([timestamp, temp_val, ppm_val, level_val, anomaly])
00243
00244             # Data update
00245             dt = datetime.strptime(timestamp, "%Y-%m-%d %H:%M:%S")
00246             self.times.append(dt)
00247             self.temps.append(float(temp_val) if temp_val != "N/A" else None)
00248             self.gas_values.append(ppm_val)
00249             self.levels.append(float(level_val) if level_val != "N/A" else None)
00250
00251             # Maintain buffer size
00252             max_len = 100
00253             if len(self.times) > max_len:
00254                 self.times = self.times[-max_len:]
00255                 self.temps = self.temps[-max_len:]
00256                 self.gas_values = self.gas_values[-max_len:]
00257                 self.levels = self.levels[-max_len:]
00258
00259             time.sleep(0.5)
00260
00261
00262     def update_plots(self):
00263         for ax in self.axs:
00264             ax.clear()
00265
00266         # --- Temperature ---
00267         if self.times and any(t is not None for t in self.temps):
00268             temps_clean = [t if t is not None else float('nan') for t in self.temps]
00269             self.axs[0].plot(self.times, temps_clean, 'r-', label='Temperature (°C)')
00270         self.axs[0].axhline(self.temp_threshold.get(), color='r', linestyle='--', label='Temp
    Threshold')
00271         self.axs[0].set_ylabel("Temperature (°C)")
00272         self.axs[0].legend(loc='upper right')
00273         self.axs[0].grid(True)
00274
00275         # --- Gas PPM ---
00276         if self.times:
00277             self.axs[1].step(self.times, self.gas_values, 'g-', label='Gas PPM')
00278         self.axs[1].axhline(self.gas_threshold.get(), color='g', linestyle='--', label='Gas
    Threshold')
00279         self.axs[1].set_ylabel("Gas PPM")
00280         self.axs[1].set_ylim(-0.1, 1.1)
00281         self.axs[1].legend(loc='upper right')
00282         self.axs[1].grid(True)
00283
00284         # --- Ultrasonic Level ---
00285         if self.times and any(l is not None for l in self.levels):
00286             levels_clean = [l if l is not None else float('nan') for l in self.levels]
00287             self.axs[2].plot(self.times, levels_clean, 'b-', label='Level (cm)')
00288         self.axs[2].set_ylabel("Level (cm)")
00289         self.axs[2].set_xlabel("Time")
00290         self.axs[2].legend(loc='upper right')
00291         self.axs[2].grid(True)
00292
```

```
00293         self.fig.autofmt_xdate()
00294         self.canvas.draw()
00295 
00296         # Fault detection and GUI update
00297         fault_msg = ""
00298         if self.temps and self.gas_values and self.levels:
00299             last_temp = self.temps[-1]
00300             last_gas = self.gas_values[-1]
00301             last_level = self.levels[-1]
00302 
00303             if last_temp is None or last_level is None:
00304                 fault_msg = "Critical fault: Sensor data missing!"
00305             elif last_temp > self.temp_threshold.get() or last_gas > self.gas_threshold.get() or not
      (0 <= last_level <= 400):
00306                 fault_msg = "Critical fault detected!"
00307 
00308         self.status_label.config(text=fault_msg, foreground='red' if fault_msg else 'green')
00309         self.root.after(1000, self.update_plots)
00310 
00311 
00312     def on_close(self):
00313         GPIO.cleanup()
00314         self.root.destroy()
00315 
00316 
00317 # ================================================================
00318 #                   MAIN PROGRAM ENTRY POINT
00319 # ================================================================
00320 
00321 
00322 if __name__ == "__main__":
00323     root = tk.Tk()
00324     app = SensorApp(root)
00325     root.mainloop()
```