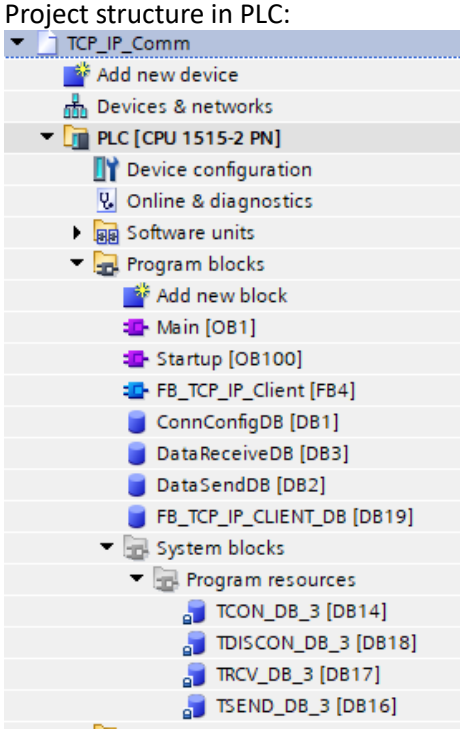
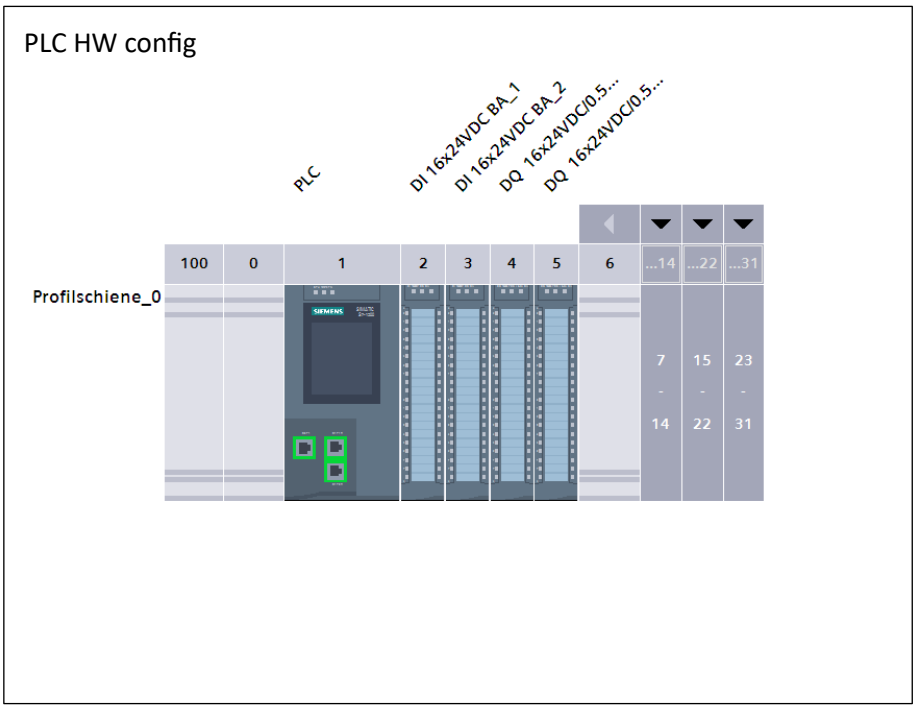
Server-Client TCP/IP (Socket) communication

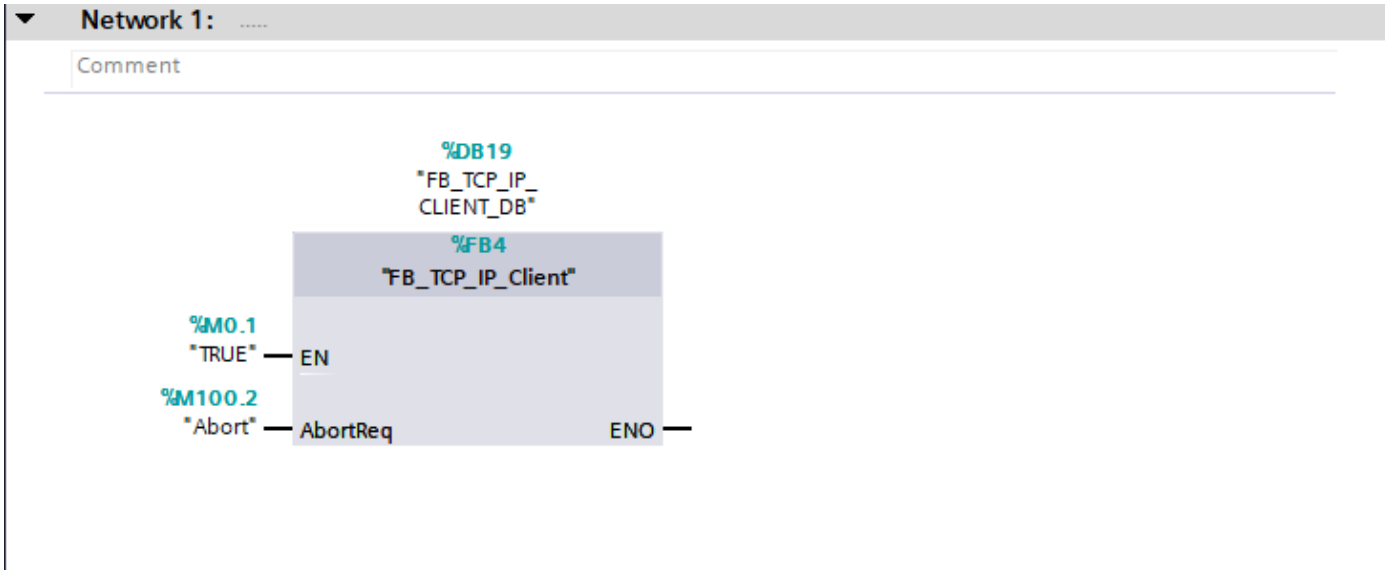
Project for TCP/IP communication between PLC (Siemens S7-1500) - as client and Python App – as server.
PLC acts here as gateway which transmits the PLC hardware inputs status to the external application and reads controls for PLC hardware outputs from the same application.

Project structure in PLC:





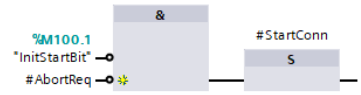
OB1:



FB_TCP_IP_Client[FB4]:

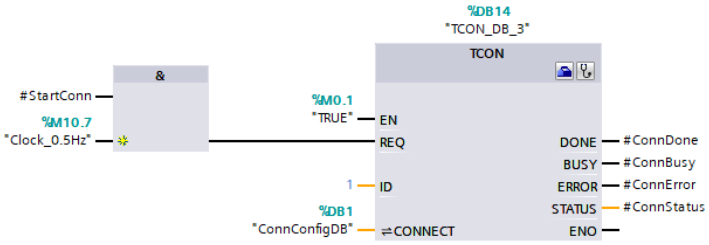
Network 1: Start Condition

Comment



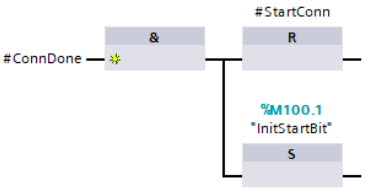
Network 2: Configure Connection FB TCON

Comment



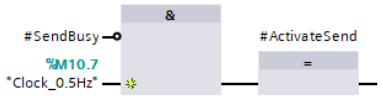
Network 3: Disable StartConn when TCON successfully connected

Comment



Network 4: Trigger for Sending Data by TSEND

Comment



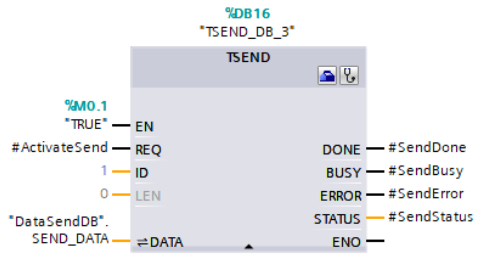
Network 5: Mapping of HW inputs to SEND_DATA array

Comment

```
1  "DataSendDB".SEND_DATA[0] := "IB1";
2  "DataSendDB".SEND_DATA[1] := "IB0";
3  "DataSendDB".SEND_DATA[2] := "IB3";
4  "DataSendDB".SEND_DATA[3] := "IB2";
```

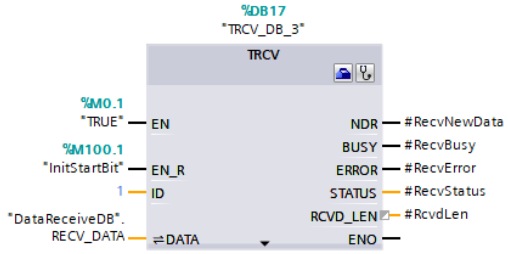
Network 6: TCP IP Client Send data

Comment



Network 7: TCP IP Client receive data

Comment



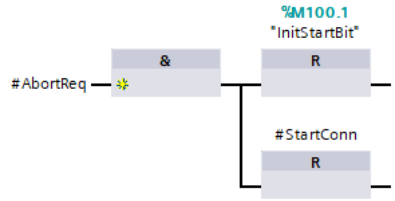
Network 8: Mapping of RECV_DATA array to HW outputs

Comment

```
1  "QB0" := "DataReceiveDB".RECV_DATA[1];
2  "QB1" := "DataReceiveDB".RECV_DATA[0];
3  "QB2" := "DataReceiveDB".RECV_DATA[3];
4  "QB3" := "DataReceiveDB".RECV_DATA[2];
```

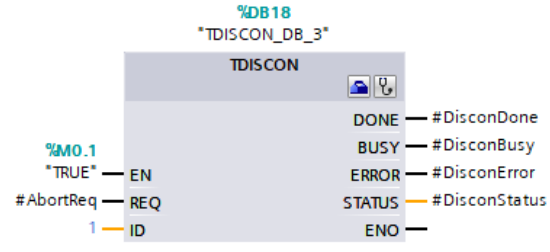
Network 9: Stop communication handler

Comment

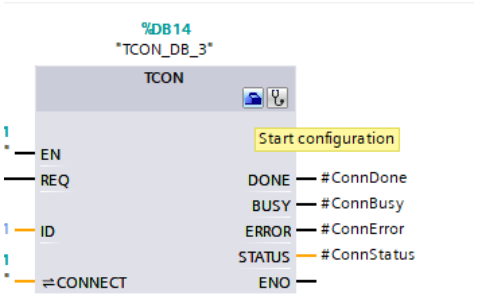


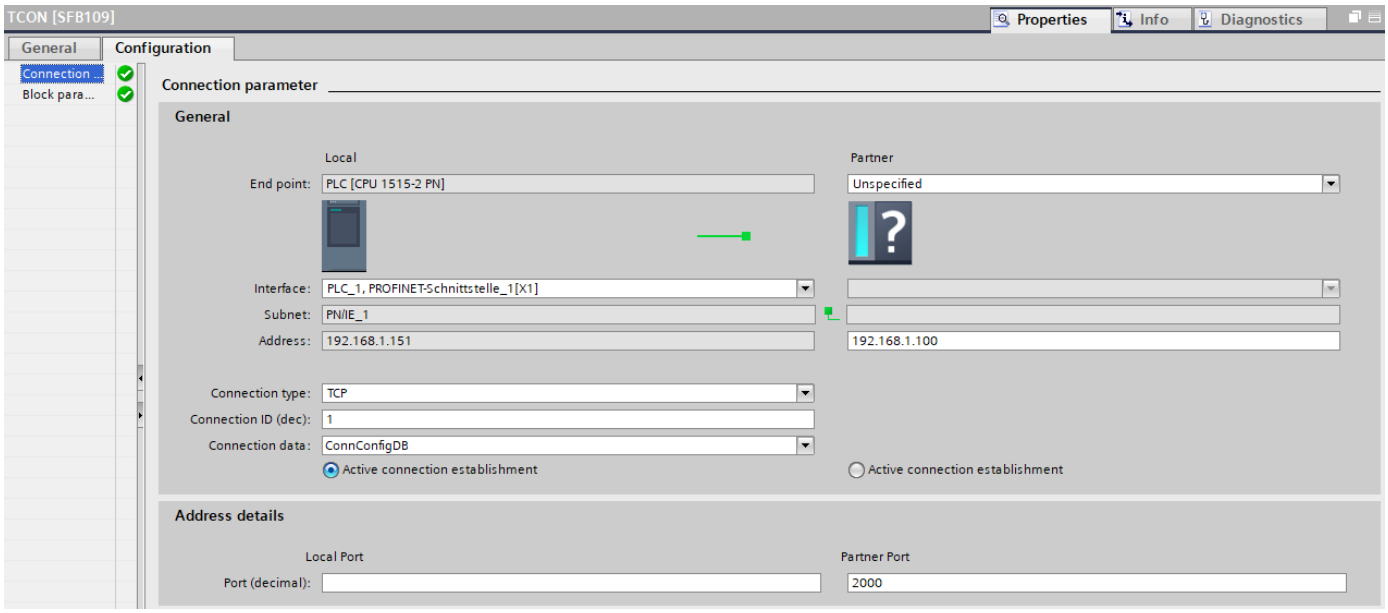
Network 10: TCP IP Client disconnect

Comment

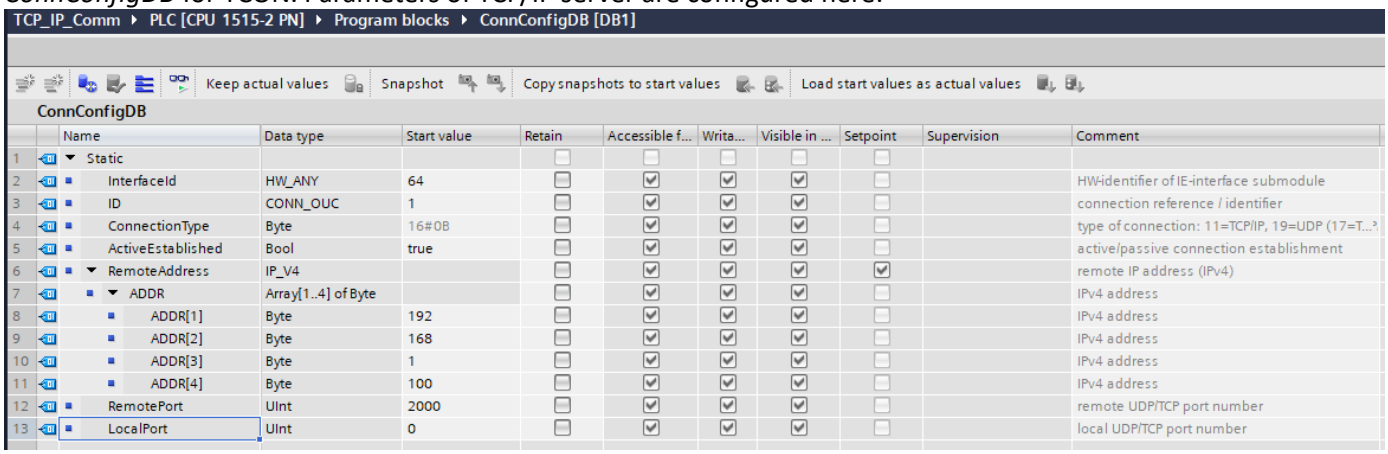


TCON in network 2 must be properly configured.
For partner settings put IP addres and port of the TCP/IP server.



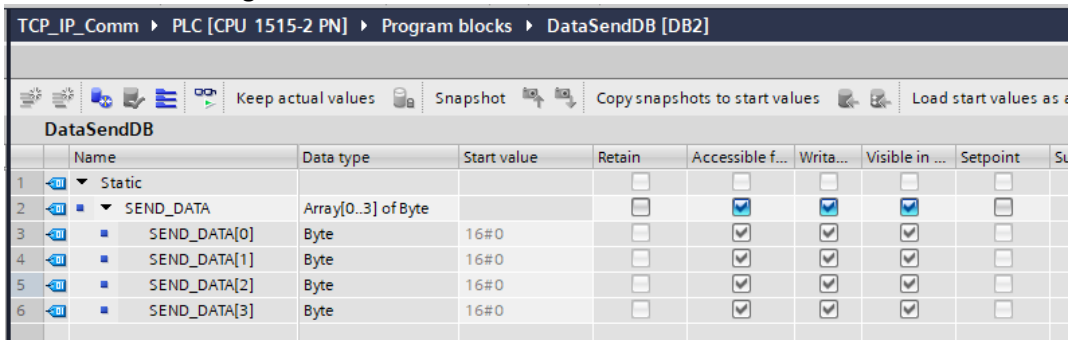


ConnConfigDB for TCON. Parameters of TCP/IP server are configured here:



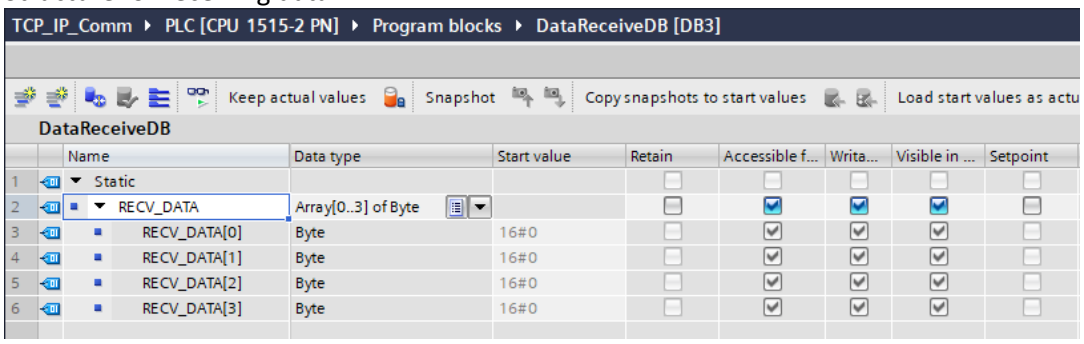
	Name	Data type	Start value	Retain	Accessible f...	Writa...	Visible in ...	Setpoint	Supervision	Comment
1	Static									
2	InterfaceId	HW_ANY	64		✓	✓	✓			HW-identifier of IE-interface submodule
3	ID	CONN_OUC	1		✓	✓	✓			connection reference / identifier
4	ConnectionType	Byte	16#0B		✓	✓	✓			type of connection: 11=TCP/IP, 19=UDP (17=T...)
5	ActiveEstablished	Bool	true		✓	✓	✓			active/passive connection establishment
6	RemoteAddress	IP_V4			✓	✓	✓	✓		remote IP address (IPv4)
7	ADDR	Array[1..4] of Byte			✓	✓	✓			IPv4 address
8	ADDR[1]	Byte	192		✓	✓	✓			IPv4 address
9	ADDR[2]	Byte	168		✓	✓	✓			IPv4 address
10	ADDR[3]	Byte	1		✓	✓	✓			IPv4 address
11	ADDR[4]	Byte	100		✓	✓	✓			IPv4 address
12	RemotePort	UInt	2000		✓	✓	✓			remote UDP/TCP port number
13	LocalPort	UInt	0		✓	✓	✓			local UDP/TCP port number

Structure for sending data



	Name	Data type	Start value	Retain	Accessible f...	Writa...	Visible in ...	Setpoint
1	Static							
2	SEND_DATA	Array[0..3] of Byte			✓	✓	✓	
3	SEND_DATA[0]	Byte	16#0		✓	✓	✓	
4	SEND_DATA[1]	Byte	16#0		✓	✓	✓	
5	SEND_DATA[2]	Byte	16#0		✓	✓	✓	
6	SEND_DATA[3]	Byte	16#0		✓	✓	✓	

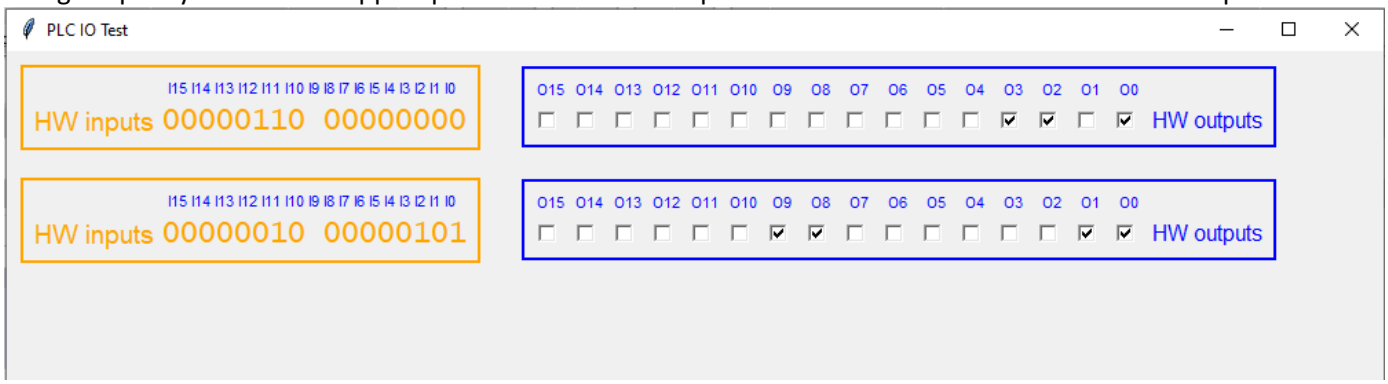
Structure for receiving data:



Inputs and outputs addressing:

	IB0	Standard-Variablen...	Byte	%IB0
	IB1	Standard-Variablen...	Byte	%IB1
	IB2	Standard-Variablen...	Byte	%IB2
	IB3	Standard-Variablen...	Byte	%IB3
	QB0	Standard-Variablen...	Byte	%QB0
	QB1	Standard-Variablen...	Byte	%QB1
	QB2	Standard-Variablen...	Byte	%QB2
	QB3	Standard-Variablen...	Byte	%QB3

Using simple Python Tkinter app its possible to read HW inputs from PLC and send controls for HW outputs in PLC:



Pyhon code:

```
""" Simple GUI app to test TCP/IP socket communication with a client device.
    The app creates a TCP/IP server that listens for incoming connections from a client.
"""
import socket
import tkinter as tk
import threading
import queue

HOST = "192.168.1.100"  # TCP/IP server address
PORT = 2000  # Port to listen on (non-privileged ports are > 1023)

# queues for data exchange between threads
q_send_to_client = queue.Queue(maxsize=4)
q_read_from_client = queue.Queue(maxsize=4)

class Backend(threading.Thread):
    """ Thread to handle TCP/IP socket communication with the client device """
    def __init__(self):
        super().__init__()

        with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
            s.bind((HOST, PORT))
            s.listen()
            conn, addr = s.accept()
            with conn:
                print(f"Connected by {addr}")

                while True:
                    data = conn.recv(4)
                    data = data.hex(':')
                    data = list(data.split(':'))
                    result = [int(idx, 16) for idx in data]

                    print("Received raw data: ", data)
                    print("Decimal data", result)

                    if not data:
                        break

                    # put received data into queue for GUI thread
                    if not q_read_from_client.full():
                        q_read_from_client.put_nowait(result)
```

```
                    if not q_send_to_client.empty():

                        data_to_client = q_send_to_client.get_nowait()
                        # encapsulate MSB & LSB from two 16-bit integers
                        msb_hw_out1 = (data_to_client[0] >> 8) & 0xFF
                        lsb_hw_out1 = data_to_client[0] & 0xFF
                        msb_hw_out2 = (data_to_client[1] >> 8) & 0xFF
                        lsb_hw_out2 = data_to_client[1] & 0xFF

                        conn.sendall(bytes([msb_hw_out1, lsb_hw_out1, msb_hw_out2, lsb_hw_out2]))

class App(tk.Tk):
    """ Main TKinter GUI application class """
    def __init__(self):
        super().__init__()
        self.out1 = 0
        self.outputs1_cb_values = []
        self.out2 = 0
        self.outputs2_cb_values = []

        self.inputs1_cb_values = []
        self.in1 = 0

        self.frame_out1 = tk.Frame(self, bd=5, height=150, width=100,
                                   highlightthickness=2, highlightbackground='blue')
        self.frame_out2 = tk.Frame(self, bd=5, height=80, width=50,
                                   highlightthickness=2, highlightbackground='blue')
        self.frame_in1 = tk.Frame(self, bd=5, height=150, width=100,
                                  highlightthickness=2, highlightbackground='orange')
        self.frame_in2 = tk.Frame(self, bd=5, height=150, width=100,
                                  highlightthickness=2, highlightbackground='orange')

        self.frame_out1.grid(row=0, column=2, padx=20, pady=10)
        self.frame_out2.grid(row=1, column=2, padx=20, pady=10)
        self.frame_in1.grid(row=0, column=0, padx=10, pady=10)
        self.frame_in2.grid(row=1, column=0, padx=10, pady=10)

        self.geometry('1000x400')
        self.title('PLC IO Test')

        # create HW outputs checkboxes
        for idx in range(15, -1, -1):
            self.outputs1_cb_values.append(tk.IntVar(value=0))
            tk.Label(self.frame_out1, text="O" + str(15-idx),
                     font=("Arial", 8),
```

```
                fg='blue').grid(row=0, column=idx)

        tk.Checkbutton(self.frame_out1,
                       state='normal',
                       variable=self.outputs1_cb_values[15-idx]).grid(row=1, column=idx)

        self.outputs2_cb_values.append(tk.IntVar(value=0))
        tk.Label(self.frame_out2, text="O" + str(15-idx),
                 font=("Arial", 8),
                 fg='blue').grid(row=0, column=idx)

        tk.Checkbutton(self.frame_out2,
                       state='normal',
                       variable=self.outputs2_cb_values[15-idx]).grid(row=1, column=idx)

    tk.Label(self.frame_out1, text="HW outputs", font=20, fg='blue').grid(row=1, column=16)
    tk.Label(self.frame_out2, text="HW outputs", font=20, fg='blue').grid(row=1, column=16)

    # create HW inputs checkboxes
    txt = ''
    for idx in range(0, 16, 1):
        txt = txt + "I" + str(15-idx) + ' '

    tk.Label(self.frame_in1, text=txt, font=("Arial", 8), fg='blue').grid(row=0, column=1)
    tk.Label(self.frame_in2, text=txt, font=("Arial", 8), fg='blue').grid(row=0, column=1)

    tk.Label(self.frame_in1, text="HW inputs",
                          font=("Arial", 14),
                          fg='orange').grid(row=1, column=0)
    tk.Label(self.frame_in2, text="HW inputs",
                          font=("Arial", 14),
                          fg='orange').grid(row=1, column=0)

def update_inputs(self):
    """ Read inputs from queue and update the HW inputs display """
    if not q_read_from_client.empty():
        get_data = q_read_from_client.get_nowait()

        text_var_in1 = tk.StringVar()
        text_var_in1.set(f"""{format(get_data[0], '#010b')[2:]} {format(get_data[1], '#010b')[2:]}""")

        byte_inputs_1 = tk.Label(self.frame_in1,
                                 textvariable=text_var_in1,
                                 font=("Lucida Console", 16), fg='orange')
        byte_inputs_1.grid(row=1, column=1)
```

```
            text_var_in2 = tk.StringVar()
            text_var_in2.set(f"""{format(get_data[2], '#010b')[2:]} {format(get_data[3], '#010b')[2:]}""")

            byte_inputs_2 = tk.Label(self.frame_in2,
                                     textvariable=text_var_in2,
                                     font=("Lucida Console", 16), fg='orange')
            byte_inputs_2.grid(row=1, column=1)

        self.after(100, self.update_inputs)

    def update_outputs(self):
        """ Read outputs from checkboxes and
            update the output data to send to TCP/IP client
        """
        self.out1 = 0
        self.out2 = 0

        # for idx in range(len(self.outputs1_cb_values)):
        #     value = self.outputs1_cb_values[idx].get()
        #     if value==1:
        #         self.out1 |= (1 << idx)

        for idx, output_cb in enumerate(self.outputs1_cb_values):
            value = self.outputs1_cb_values[idx].get()
            if value==1:
                self.out1 |= (1 << idx)

        for idx in range(len(self.outputs2_cb_values)):
            value = self.outputs2_cb_values[idx].get()
            if value==1:
                self.out2 |= (1 << idx)

        if not q_send_to_client.full():
            q_send_to_client.put_nowait([self.out1, self.out2])

        self.after(200, self.update_outputs)

if __name__ == "__main__":

    threading.Thread(target=Backend, daemon=True).start()

    app = App()
    app.after(200, app.update_outputs)
    app.after(100, app.update_inputs)
    app.mainloop()
```