IEEE IMFW 2026

Student Filter Tuning Competition

S2P Data Acquisition Guide v1.0

Contents

1. Overview

This document provides detailed information about the Vector Network Analyzer (VNA) model used in the IMFW 2026 Student Filter Tuning Competition, along with a sample Python script for acquiring S parameters data. It serves as a reference for participants who will be using their self-developed computer-aided tuning (CAT) tools.

2. Prerequisite

- Keysight IO Libraries Suite must be installed.
- Instrument access via TCP/IP (VISA)

2. Instruments

2.1 Keysight E5080A

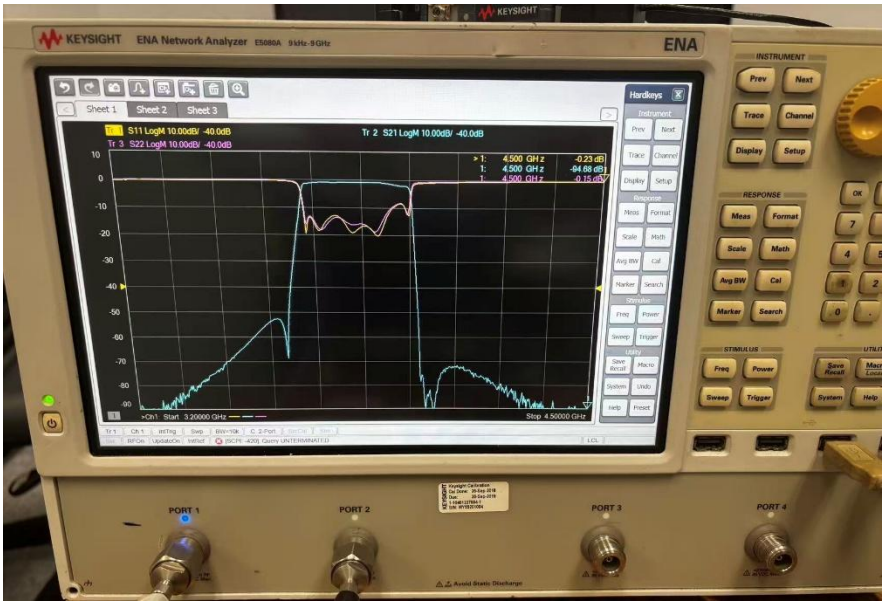

Figure 1 E5080A

2.2 Agilent E5071B

Figure 2 E5071B

3. Measurement and Data Acquisition Logic

3.1 Common Configuration

Both scripts share the following configuration principles:

- VISA communication over TCP/IP
- Binary data format: `REAL,64`
- Little-endian byte order (`FORM:BORD SWAP`)
- Single-sweep synchronization using `*OPC?`

3.2 Data Integrity

- Frequency data and S-parameter data are explicitly aligned
- Real and imaginary components are preserved without magnitude/phase conversion
- Output strictly follows Touchstone S2P conventions

4. Data Acquisition Python Script

4.1 Keysight E5080A

```
import pyvisa
import sys
import os

# ========================================
# E5080A VNA Data Reader
# ========================================

def main():
  # VNA IP Address
  address = "TCPIP0::172.17.2.2::inst0::INSTR"
  try:
    rm = pyvisa.ResourceManager()
    print(f"Connecting to {address}...")
    vna = rm.open_resource(address)
    vna.timeout = 60000  # 60 seconds timeout
    vna.clear()
    idn = vna.query("*IDN?")
    print(f"Connected: {idn.strip()}")
    print("Configuring data format to REAL,64 (Binary)...")
    vna.write(":FORMat:DATA REAL,64")
    vna.write(":FORMat:BORDer SWAP")
    print("Fetching S2P data using :CALCulate:MEASure:DATA:SNP? ...")
    values = vna.query_binary_values(':CALCulate:MEASure:DATA:SNP?', datatype='d',
is_big_endian=False)
    if not values:
      print("Error: No data received.")
      return
    print(f"Received {len(values)} data points.")
    raw_output_file = "raw_data.txt"
    print(f"Saving raw values to {raw_output_file} for inspection...")
    with open(raw_output_file, "w") as f_raw:
      for idx, val in enumerate(values):
        f_raw.write(f"{idx}: {val}\n")
    print(f"Saved raw data to {os.path.abspath(raw_output_file)}")
    num_points = len(values) // 9
    print(f"Detected {num_points} frequency points.")
    freqs = values[:num_points]
    s_data = values[num_points:]
    print(f"First 3 Frequencies: {freqs[:3]}")
    num_params = 8
    if len(s_data) != num_points * num_params:
      print(f"Warning: Data length {len(s_data)} does not match expected {num_points *
num_params}")
    print("Reformatting data (Transposing blocks)...")
    print("Saving to file...")
    output_file = "data_e5080.s2p"
```

```
        with open(output_file, "w") as f:
            # Write Header
            f.write("! S2P File generated by Python script\n")
            f.write("# Hz S RI R 50\n") # Frequency in Hz, S-parameter, Real-Imaginary, 50 Ohm
            f.write("! Freq\tReS11\tImS11\tReS21\tImS21\tReS12\tImS12\tReS22\tImS22\n")
            for i in range(num_points):
                # Get Frequency
                freq = freqs[i]
                point_values = []
                for p in range(num_params):
                    val_index = p * num_points + i
                    if val_index < len(s_data):
                        point_values.append(s_data[val_index])
                    else:
                        point_values.append(0.0) # Should not happen
                line = f"{freq:.6f}   " + "   ".join(f"{val:.6f}" for val in point_values) + "\n"
                f.write(line)
                if i == 0:
                    print(f"Preview - First Point Data:\n{line.strip()}")
        print(f"Success! S2P data saved to {os.path.abspath(output_file)}")
        vna.close()
        rm.close()
    except pyvisa.VisaIOError as e:
        print(f"VISA Error: {e}")
    except Exception as e:
        print(f"Error: {e}")

if __name__ == "__main__":
    main()
```

4.2 Keysight E5071B

```
import pyvisa
import time
import struct
import os

# ========================================
# E5071B VNA Data Reader
# ========================================

def main():

  ip_address = "172.17.2.2"
  visa_address = f"TCPIP0::{ip_address}::inst0::INSTR"
  channel = 1
  timeout_ms = 20000
  try:
    rm = pyvisa.ResourceManager()
    print(f"Connecting to {visa_address}...")
    vna = rm.open_resource(visa_address)
    vna.timeout = timeout_ms
    vna.read_termination = '\n'
    vna.write_termination = '\n'
    idn = vna.query("*IDN?")
    print(f"Connected: {idn.strip()}")
    print("Initializing measurement...")
    vna.write("*CLS")  # Clear status
    vna.write(f":DISP:WIND{channel}:ACT")  # Activate measurement window
    print("Configuring traces to S11, S21, S12, S22...")
    vna.write(f":CALC{channel}:PAR:COUN 4")
    trace_cfg = {1: "S11", 2: "S21", 3: "S12", 4: "S22"}
    for t_id, param in trace_cfg.items():
      vna.write(f":CALC{channel}:PAR{t_id}:DEF {param}")
      vna.write(f":CALC{channel}:PAR{t_id}:SEL")
    print("Triggering single sweep...")
    vna.write(":ABOR")  # Abort any ongoing measurement
    vna.write(f":INIT{channel}:CONT ON")  # Continuous measurement off
    vna.write("TRIG:SEQ:SCOP ACT")  # Trigger scope to active channel
    vna.write(":TRIG:SOUR MAN")  # Set trigger source to manual
    vna.write(":TRIG:SING")  # Trigger a single sweep
    # Wait for operation complete using *OPC? query
    print("Waiting for sweep to complete (*OPC?)...")
    vna.query("*OPC?")
    # Set data format to binary real numbers (64-bit floating point)
    print("Configuring data format...")
    vna.write("FORM:DATA REAL")  # REAL,64 (8 bytes per value)
    vna.write("FORM:BORD SWAP")  # SWAP = Little Endian (PC format)
    print("Reading frequency data...")
    freqs = vna.query_binary_values(f":SENS{channel}:FREQ:DATA?",
                    datatype='d',
```

```
                    is_big_endian=False)
  num_points = len(freqs)
  print(f"Number of points: {num_points}")
  print("Reading S-parameter data (using SDAT per screenshot)...")
  param_data = {}
  for tr in range(1, 5):
    try:
      vna.write(f":CALC{channel}:PAR{tr}:SEL")
      param_def = vna.query(f":CALC{channel}:PAR{tr}:DEF?")
      param_name = param_def.strip()
      print(f"Trace {tr} is {param_name}")
      vals = vna.query_binary_values(f":CALC{channel}:DATA:SDAT?",
                  datatype='d',
                  is_big_endian=False)
      c_vals = []
      for k in range(0, len(vals), 2):
        c_vals.append((vals[k], vals[k+1]))
      param_data[param_name] = c_vals
    except Exception as e:
      print(f"Could not read Trace {tr}: {e}")
  output_file = "data_e5071.s2p"
  print(f"Saving to {output_file}...")
  with open(output_file, "w") as f:
    # Write S2P file header
    f.write("! S2P File generated by Python script (E5071)\n")
    f.write("# Hz S RI R 50\n")  # Frequency in Hz, S-params, Real/Imag, 50 Ohm reference
    f.write("! Freq  ReS11  ImS11  ReS21  ImS21  ReS12  ImS12  ReS22  ImS22\n")
    def get_val(p_name, idx):
      if p_name in param_data and idx < len(param_data[p_name]):
        return param_data[p_name][idx]
      return (0.0, 0.0)
    for i in range(num_points):
      freq = freqs[i]
      s11 = get_val("S11", i)
      s21 = get_val("S21", i)
      s12 = get_val("S12", i)
      s22 = get_val("S22", i)
      line = f"{freq:.6f}  " \
          f"{s11[0]:.6f}  {s11[1]:.6f}  " \
          f"{s21[0]:.6f}  {s21[1]:.6f}  " \
          f"{s12[0]:.6f}  {s12[1]:.6f}  " \
          f"{s22[0]:.6f}  {s22[1]:.6f}\n"
      f.write(line)
      if i == 0:
        print(f"Preview - First Point:\n{line.strip()}")

  print(f"Done! Saved to {os.path.abspath(output_file)}")
  vna.write("FORM:DATA ASC")
  vna.close()
  rm.close()
except pyvisa.VisaIOError as e:
  print(f"VISA Error: {e}")
```

```
    except Exception as e:
        print(f"Error: {e}")
        import traceback
        traceback.print_exc()

if __name__ == "__main__":
    main()
```