```
Private Sub Label6_Click()

End Sub

Private Sub Label7_Click()

End Sub

Private Sub Label8_Click()

End Sub

Private Sub Label9_Click()

End Sub

Private Sub ListBox1_Click()

End Sub

Private Sub MultiPage1_Change()

End Sub

Private Sub ScrollBar1_Change()

End Sub

Private Sub ScrollBar2_Change()

End Sub

Private Sub SpinButton1_Change()

End Sub

Private Sub TabStrip1_Change()

End Sub

Private Sub TextBox10_Change()

End Sub

Private Sub TextBox11_Change()

End Sub

Private Sub TextBox12_Change()

End Sub

Private Sub TextBox14_Change()

End Sub

Private Sub TextBox16_Change()

End Sub

Private Sub TextBox17_Change()

End Sub

Private Sub TextBox2_Change()

End Sub

Private Sub TextBox3_Change()

End Sub

Private Sub TextBox4_Change()
```

```
End Sub

Private Sub TextBox6_Change()

End Sub

Private Sub TextBox7_Change()

End Sub

Private Sub TextBox8_Change()

End Sub

Private Sub TextBox9_Change()

End Sub

Private Sub ToggleButton1_Click()

End Sub

Private Sub UserForm_Click()

End Sub
import numpy as np
from scipy.fft import fft
from scipy.signal import butter, filtfilt, hilbert

def capture_signal(fs = 1000):
    t = np.linspace(0, 1, fs)
    signal = np.Sin(2 * np.pi * 50 * t) + 0.5 * np.random.randn(Len(t))
    return t, signal

def apply_fft(signal):
    return fft(signal)

def calculate_snr(signal, noise_level = 0.5):
    power_signal = np.mean(signal**2)
    power_noise = noise_level**2
    return 10 * np.log10(power_signal / power_noise)

def classify_bandwidth(lowcut = 40, highcut = 60):
    bandwidth = highcut - lowcut
    return "Narrowband" if bandwidth < 30 else "Broadband"

def check_linearity(signal):
    second_derivative = np.gradient(np.gradient(signal))
    return "Linear" if np.allclose(second_derivative, 0, atol=0.01) else "Nonlinear"

def preprocess_signal(signal):
    b, a = butter(4, [0.05, 0.95], btype='band')
    filtered = filtfilt(b, a, signal)
    rectified = np.Abs(filtered)
    return rectified

def detect_modulation(signal):
    analytic_signal = hilbert(signal)
    amplitude_envelope = np.Abs(analytic_signal)
    Phase = np.unwrap(np.Angle(analytic_signal))
    return amplitude_envelope, phase

# Execution
t , raw_signal = capture_signal()
clean_signal = preprocess_signal(raw_signal)
fft_signal = apply_fft(clean_signal)
snr = calculate_snr(clean_signal)

if snr < 10:
    classification = "Noise"
Else:
    bandwidth_type = classify_bandwidth()
```

```
    linearity = check_linearity(clean_signal)
    classification = f"{bandwidth_type}, {linearity}"

amplitude , Phase = detect_modulation(clean_signal)

print("Signal Classification:", classification)
print("Modulation Envelope Sample:", amplitude[:5])


'
' ent Macro
'
'
```

```
   Print
[Apply Fourier Transform]
   Print
[Evaluate Signal-to-Noise Ratio (SNR)]
   ??? SNR < Threshold ? [Classify as Noise (Signal Bruit)] ? [Log & Discard]
   ??? SNR ? Threshold
         Print
[Check Bandwidth]
   ??? Bandwidth < ? ? [Classify as Narrowband Signal (Signal Bande Étroite)]
   ??? Bandwidth ? ? ? [Classify as Broadband Signal]
         Print
[Check Linearity]
   ??? d²y/dx² ? 0 ? [Classify as Linear Signal]
   ??? d²y/dx² ? 0 ? [Apply Curve Fitting or Nonlinear Analysis]
         Print
[Store Signal Metadata + Visualization]
   Print
End
?? Algorigramme (Algorithmic Flow)
python
import numpy as np
from scipy.fft import fft
from scipy.signal import butter, filtfilt

def capture_signal():
    t = np.linspace(0, 1, 1000)
    signal = np.Sin(2 * np.pi * 50 * t) + 0.5 * np.random.randn(Len(t))
    return signal

def apply_fft(signal):
    return fft(signal)

def calculate_snr(signal, noise_level = 0.5):
    power_signal = np.mean(signal**2)
    power_noise = noise_level**2
    return 10 * np.log10(power_signal / power_noise)

def classify_bandwidth(signal, fs = 1000, lowcut = 40, highcut = 60):
    bandwidth = highcut - lowcut
    return "Narrowband" if bandwidth < 30 else "Broadband"

def check_linearity(signal):
    second_derivative = np.gradient(np.gradient(signal))
    return "Linear" if np.allclose(second_derivative, 0, atol=0.01) else "Nonlinear"

# Execution
signal = capture_signal()
fft_signal = apply_fft(signal)
snr = calculate_snr(signal)

if snr < 10:
    classification = "Noise"
Else:
    bandwidth_type = classify_bandwidth(signal)
    linearity = check_linearity(signal)
```

```
    classification = f"{bandwidth_type}, {linearity}"

print("Signal Classification:", classification)
   Print
[Capture Raw Signal or Image]
   Print
[Apply Preprocessing]
   ??? Filter (Butterworth, Monochromatic)
   ??? Normalize & Rectify (Redresseur)
   ??? Denoise (Image Bruit, Noyaux)
   Print
[Signal Analysis]
   ??? Fourier Transform ? Frequency Domain
   ??? SNR Evaluation ? Signal Bruit Filtering
   ??? Bandwidth Check ? Narrowband/Broadband
   ??? Dispersion Analysis ? Group Delay
   ??? Linearity/Colinearity ? d²y/dx²
   Print
[Modulation Logic]
   ??? Detect Modulation Type (AM/FM/PM)
   ??? Apply Demodulation
   ??? Multiplex/Scale (Time/Frequency Division)
   Print
[Control Logic]
   ??? Transfer Function Modeling (Nichol Chart)
   ??? Oscillation Detection
   ??? Interval Analysis (Finite/Infinite)
   Print
[Code/Decode Logic]
   ??? Encode Signal Metadata
   ??? Decode for LMS/Thesis Integration
   Print
[Store + Visualize]
   ??? GitHub CI/CD Logs
   ??? LMS Module Outputs
   ??? AIU Thesis Artifacts
   Print
End
?? Python-Based Algorigramme (Modular Diagnostic Flow)
python
from scipy.fft import fft
from scipy.signal import butter, filtfilt, hilbert
import numpy as np

def preprocess_signal(signal, fs = 1000):
    b, a = butter(4, [0.05, 0.95], btype='band')
    filtered = filtfilt(b, a, signal)
    rectified = np.Abs(filtered)
    return rectified

def detect_modulation(signal):
    analytic_signal = hilbert(signal)
    amplitude_envelope = np.Abs(analytic_signal)
    instantaneous_phase = np.unwrap(np.Angle(analytic_signal))
    return amplitude_envelope, instantaneous_phase

def nichol_chart_transfer(signal):
    # Placeholder for transfer function modeling
    Gain = np.Max(signal) / np.Min(signal)
    phase_margin = np.angle(fft(signal)[1])
    return gain, phase_margin

def multiplex_signal(signal, Method = "TDM"):
    if method == "TDM":
        return signal[::2], signal[1::2]
    elif method == "FDM":
        return np.split(signal, 2)
    Else:
        return signal

# Execution
raw_signal = np.Sin(2 * np.pi * 50 * np.linspace(0, 1, 1000)) + 0.3 * np.random.randn(1000)
clean_signal = preprocess_signal(raw_signal)
```

```
amplitude , Phase = detect_modulation(clean_signal)
Gain , phase_margin = nichol_chart_transfer(clean_signal)
mux1 , mux2 = multiplex_signal(clean_signal)

print("Modulation Envelope:", amplitude[:5])
print("Nichol Gain:", gain, "Phase Margin:", phase_margin
[Start]
   Print
[Capture Raw Signal or Image]
   Print
[Apply Preprocessing]
   ??? Filter (Butterworth, Monochromatic)
   ??? Normalize & Rectify (Redresseur)
   ??? Denoise (Image Bruit, Noyaux)
   Print
[Signal Analysis]
   ??? Fourier Transform ? Frequency Domain
   ??? Laplace Transform ? Control Domain
   ??? SNR Evaluation ? Signal Bruit Filtering
   ??? Bandwidth Check ? Narrowband/Broadband
   ??? Dispersion Analysis ? Group Delay
   ??? Linearity/Colinearity ? d²y/dx²
   Print
[Modulation Logic]
   ??? Detect Modulation Type (AM/FM/PM)
   ??? Apply Demodulation
   ??? Multiplex/Scale (Time/Frequency Division)
   Print
[Control Logic]
   ??? Transfer Function Modeling (Nichol Chart)
   ??? Oscillation Detection
   ??? Interval Analysis (Finite/Infinite)
   ??? Component Material Evaluation (Dielectric, Conductive)
   Print
[Code/Decode Logic]
   ??? Encode Signal Metadata
   ??? Decode for LMS/Thesis Integration
   Print
[Store + Visualize]
   ??? GitHub CI/CD Logs
   ??? LMS Module Outputs
   ??? AIU Thesis Artifacts
   Print
End
?? Python-Based Algorigramme (Laplace & Transfer Logic)
import numpy as np
from scipy.signal import butter, filtfilt, TransferFunction, bode
from scipy.fft import fft

def capture_signal(fs = 1000):
    t = np.linspace(0, 1, fs)
    signal = np.Sin(2 * np.pi * 50 * t) + 0.3 * np.random.randn(fs)
    return t, signal

def preprocess(signal):
    b, a = butter(4, [0.05, 0.95], btype='band')
    filtered = filtfilt(b, a, signal)
    rectified = np.Abs(filtered)
    return rectified

def laplace_transfer(r = 1, C = 0.000001):
    # RC low-pass filter transfer function: H(s) = 1 / (RCs + 1)
    Num = [1]
    Den = [R*C, 1]
    System = TransferFunction(Num, Den)
    w , mag, Phase = bode(System)
    return w, mag, phase

def energy_integral(signal, dt = 0.001):
    return np.trapz(signal**2, dx=dt)

# Execution
t , raw_signal = capture_signal()
```

```
clean_signal = preprocess(raw_signal)
w , mag, Phase = laplace_transfer()
Energy = energy_integral(clean_signal)

print("Energy Accumulated:", energy)
print("Laplace Transfer Magnitude (first 5):", mag[:5])
?? Component & Material Design Integration
Component Type  Diagnostic Logic    Material Consideration
Transformer Core    $$ E = \int P(t)\,dt $$ Ferromagnetic saturation modeling
Antenna Array   $$ H(s) = \frac{Y(s)}{X(s)} $$  Conductivity, dispersion control
Filter Circuit  Laplace Transfer Function   Dielectric loss, bandwidth tuning
Oscillator  Phase Margin, Nichol Chart  Crystal stability, feedback gain
[Start]
   Print
[Capture Raw Signal or Image]
   Print
[Apply Preprocessing]
   ??? Filter (Butterworth, Monochromatic)
   ??? Normalize & Rectify (Redresseur)
   ??? Denoise (Image Bruit, Noyaux)
   Print
[Signal Analysis]
   ??? Fourier Transform ? Frequency Domain
   ??? Laplace Transform ? Control Domain
   ??? SNR Evaluation ? Signal Bruit Filtering
   ??? Bandwidth Check ? Narrowband/Broadband
   ??? Dispersion Analysis ? Group Delay
   ??? Linearity/Colinearity ? d²y/dx²
   Print
[Modulation Logic]
   ??? Detect Modulation Type (AM/FM/PM)
   ??? Apply Demodulation
   ??? Multiplex/Scale (Time/Frequency Division)
   Print
[Component Simulation]
   ??? Oscillator & Filter Response
   ??? Amplifier Gain Modeling
   ??? Thyristor & TRIAC Switching Logic
   ??? Condensator Charge/Discharge Curve
   ??? Oscilloscope Time-Base Simulation
   Print
[Azure ML Experimentation]
   ??? Launch Notebook for Signal Modeling

   ??? Monitor Job Status (Success/Failure)
   ??? Log Regret/Error Metrics
   Print
[Pipeline & Deployment]
   ??? Run Backtest Pipeline
   ??? Deploy Model to Real-Time Endpoint
   ??? Monitor via Kubernetes Cluster
   ??? Return Job Status & Metrics
   Print
[Code/Decode Logic]
   ??? Encode Signal Metadata
   ??? Decode for LMS/Thesis Integration
   Print
[Store + Visualize]
   ??? GitHub CI/CD Logs
   ??? LMS Module Outputs
   ??? AIU Thesis Artifacts
   Print
End
?? Azure ML + Kubernetes Integration (Python Pseudocode)
from azureml.core import Workspace, Experiment, ScriptRunConfig, Environment
from azureml.core.compute import ComputeTarget
from azureml.pipeline.core import Pipeline

# Connect to Azure ML Workspace
ws = workspace.from_config()

# Define compute cluster
cpu_cluster = ComputeTarget(workspace = ws, Name = "cpu-cluster")
```

```
# Define environment
env = environment.from_conda_specification(Name = "signal-env", file_path = "env.yml")

# Configure training job
src = ScriptRunConfig(source_directory="signal_model",
                      script="train.py",
                      compute_target=cpu_cluster,
                      environment=env)

# Launch experiment
Experiment = Experiment(workspace = ws, Name = "signal-modulation-exp")
Run = Experiment.submit(src)
Run.wait_for_completion (show_output = True)

# Check job status
Status = Run.get_status()
if status != "Completed":
    Print ("Regret: Job unsuccessful. Filing error logs.")
Else:
    Print ("Job completed successfully. Ready for deployment.")

    Print ("Regret: Job unsuccessful. Filing error logs.")
Else:
    Print ("Job completed successfully. Ready for deployment.")
```

?? Component-Level Simulation Mapping

Component Diagnostic Logic Simulation Purpose. Modulation & Demodulation Calculations

Type of Modulation	Mathematical Model	Use Case
AM (Amplitude Modulation)	$$ s(t) = [1 + m(t)] \cdot \cos(2\pi f_c t) $$	Radio broadcast, analog TV
FM (Frequency Modulation)	$$ s(t) = A \cdot \cos(2\pi f_c t + \beta \cdot \sin(2\pi f_m t)) $$	Astronomic signal encoding
PM (Phase Modulation)	$$ s(t) = A \cdot \cos(2\pi f_c t + m(t)) $$	Satellite telemetry
Demodulation	Envelope detection, PLL (Phase-Locked Loop)	Signal recovery in receivers

?? 2. Component-Level Computation

Component	Diagnostic Logic	Portfolio Integration
Oscillator	Frequency stability, waveform generation	Signal source modeling
Amplifier	Gain calculation: $$ G = \frac{V_{out}}{V_{in}} $$	Signal strength diagnostics
Thyristor/TRIAC	Switching behavior, waveform clipping	Power control simulation
Condensator	Charge/discharge: $$ V(t) = V_0 e^{-t/RC} $$	Energy storage modeling
Filter	Butterworth, Chebyshev, Monochromatic	Bandwidth shaping
Oscilloscope	Time-base visualization	Waveform inspection
Antenna Array	Radiation pattern, impedance matching	Transmission modeling

??? 3. Transmission Logic Across Domains

Domain	Signal Flow Logic	Credential Artifact
Radio Astronomy	Narrowband signal capture, dispersion modeling	Laplace-based diagnostics
Television	AM/FM modulation, video signal encoding	Component simulation logs
Telecommunication	Multiplexing (TDM/FDM), error correction	Azure ML deployment pipeline

```
Class EnergyMeter
    Public Voltage As Double
    Public Current As Double
    Public Function Power() As Double
        Power = Voltage * Current
    End Function
End Class
```

?? 2. Excel Sheet + Module + Macro Integration

Sheet Layout:

Signal Type	Value	Unit	Timestamp
Voltage	220	Volts	2025-08-29 13:51
Current	5	Amps	2025-08-29 13:51
Power	1100	Watts	Auto-calculated

Macro Example:

```
ub CalculatePower()
    Dim v As Double, i As Double
    v = Range("B2").Value
    i = Range("B3").Value
    Range("B4").Value = v * i
End Sub
```

?? 3. MS Word Project Form + Experimental Job Record

Form Elements:

" LabelText: "Energy Diagnostic Record"
" TextBox: Voltage, Current, Material Type
" Command Buttons: OK, Cancel, Next

" TabControl: Signal Input | Simulation | Credential Mapping | Export

Job Record Fields:

Field	Description
Job ID	Auto-generated unique identifier
AIU Reference	Thesis or LMS module link
Company Name	Diagnostic partner or client
Experiment Type	Signal modeling, metering, etc.
Credential Output	NQF/WA-aligned artifact

?? 4. Run Job, Record Job, Transfer Step Logic

Run Job Logic:

```
Sub RunDiagnosticJob()
    Call CalculatePower
    MsgBox "Job Completed. Power = " & Range("B4").Value & " Watts"
End Sub
```

Record Job Logic:

" Save results to Excel sheet
" Export summary to Word form
" Log CI/CD status to GitHub or LMS

Transfer Step:

" Move to next tab/page
" Trigger metering simulation
" Update credential mapping

?? 5. Metering Energy & Credential Mapping

Metering Equation: $$ E = \int_0^T P(t)\,dt $$

VBA Approximation:

```
Function EnergyMetered(PowerArray() As Double, Interval As Double) As Double
    Dim E As Double, i As Integer
    For i = LBound(PowerArray) To UBound(PowerArray)
        E = E + PowerArray(i) * Interval
    Next i
    EnergyMetered = E
End Function
```

Credential Mapping Table:

Module Name	Diagnostic Output	Credential Code
Signal Simulation	Power, Flux	NQF Level 5
Metering System	Energy Profile	WA Code 3.2.1
Job Record Form	LMS Artifact	AIU Thesis Ref

Logic Gate Testing Summary (Tasks 1-4)

Gate Type	Method	Verification
OR	Switches + Lamp	Truth table (A + B)
AND	Switches + Lamp	Truth table (A · B)

?? AND Gate Truth Table (Sample)

A	B	Voltage A	Voltage B	Output Y	LED Status
0	0	0V	0V	0	OFF
0	1	0V	5V	0	OFF
1	0	5V	0V	0	OFF
1	1	5V	5V	1	ON

? Task 4: IC 7408 Testing

" Gate 1: Pins 1, 2 ? 3
" Gate 2: Pins 4, 5 ? 6
" Gate 3: Pins 9, 10 ? 8
" Gate 4: Pins 12, 13 ? 11 ? Record outputs and verify against truth

Oscillator	Time-base & frequency stability	Signal generation for modulation
Amplificator	Gain modeling	Signal strength analysis
Thyristor/TRIAC	Switching logic	Power control simulation
Condensator	Charge/discharge curve	Energy storage modeling
Oscilloscope	Time-domain visualization	Signal waveform inspection

?? LMS & Thesis Integration Strategy

trade experimentatl continue   intelligence artificial machie zure learning for experimentation training and deployemnt

experimentation in notebooks	basic name	time
run job to traing model	name cpu cluster	filtre
run pipeline back job	job command	bandwig
deployment material	returned job	Lineare
Lacomputer cluster	radio signal	non line
Label7	capture signal	modulat multiple

ok   cancel   next