MANUAL TÉCNICO
SIPMULATOR

Desarrollado por: Sophia Pinzón[1], Javier Castaño[1], Deywis Moreno[2]

[1]Universidad Antonio Nariño
Facultad de Ingeniería Mecánica, Electrónica y Biomédica
Semillero de Investigación SITAD
Villavicencio-Colombia

[2]Universidad Antonio Nariño
Facultad de Ciencias
Bogotá-Colombia

2025

Versión 1.0

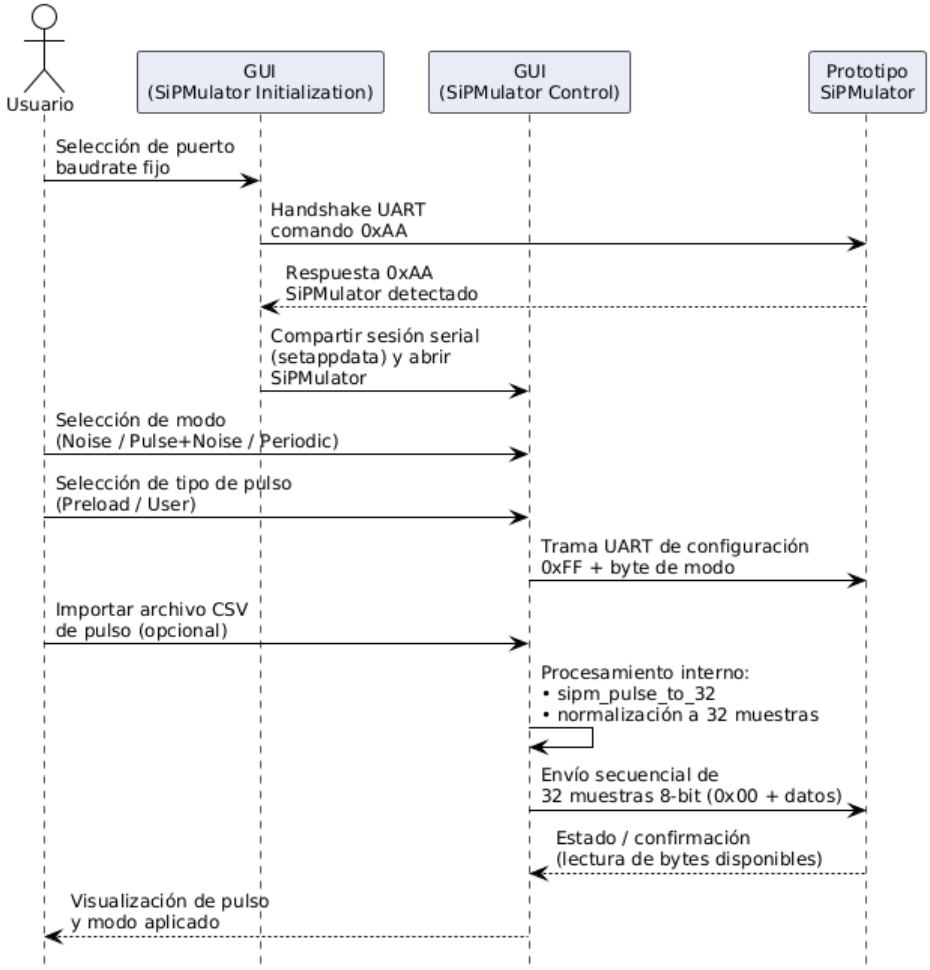
Tabla de Contenido

Pág.

1. Descripción general del sistema

SiPMulator es una aplicación desarrollada en MATLAB App Designer que permite controlar un prototipo basado en FPGA capaz de generar pulsos equivalentes a los producidos por sensores SiPM. El software gestiona la comunicación UART, la carga de pulsos, la discretización de señales, la selección del modo de salida y la visualización gráfica en tiempo real.

2. Arquitectura general del sistema



La arquitectura se divide en:

2.1. Interfaz 1: SiPMulator_Initialization

- Detección del dispositivo (comando 0xAA).
- Apertura de puerto y transferencia de sesión vía setappdata.
- Lanzamiento de SiPMulator.

2.2.Interfaz 2: SiPMulator

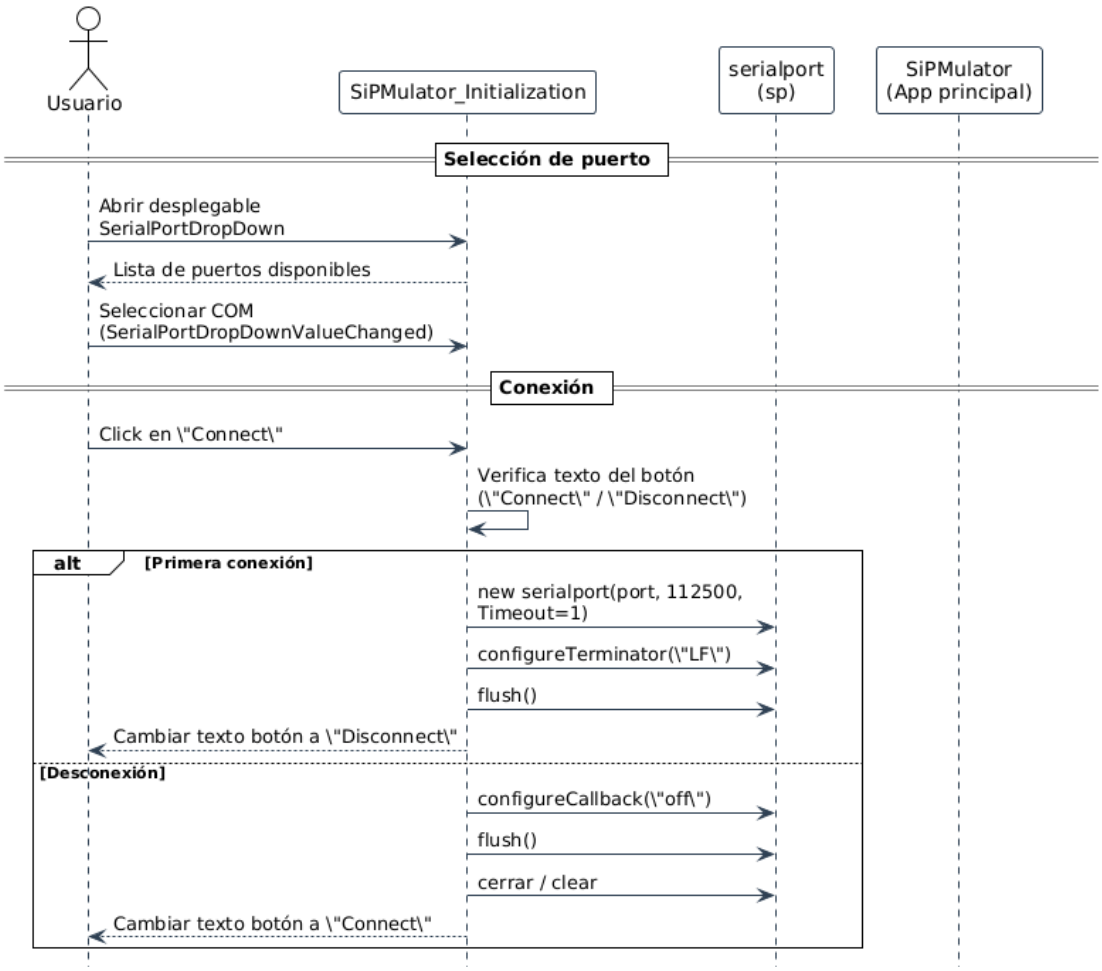
- Panel de control de señales.
- Módulo de envío de modos.
- Módulo de carga y envío de pulsos.
- Timer de verificación con FPGA.
- Renderizado en UIAxes.

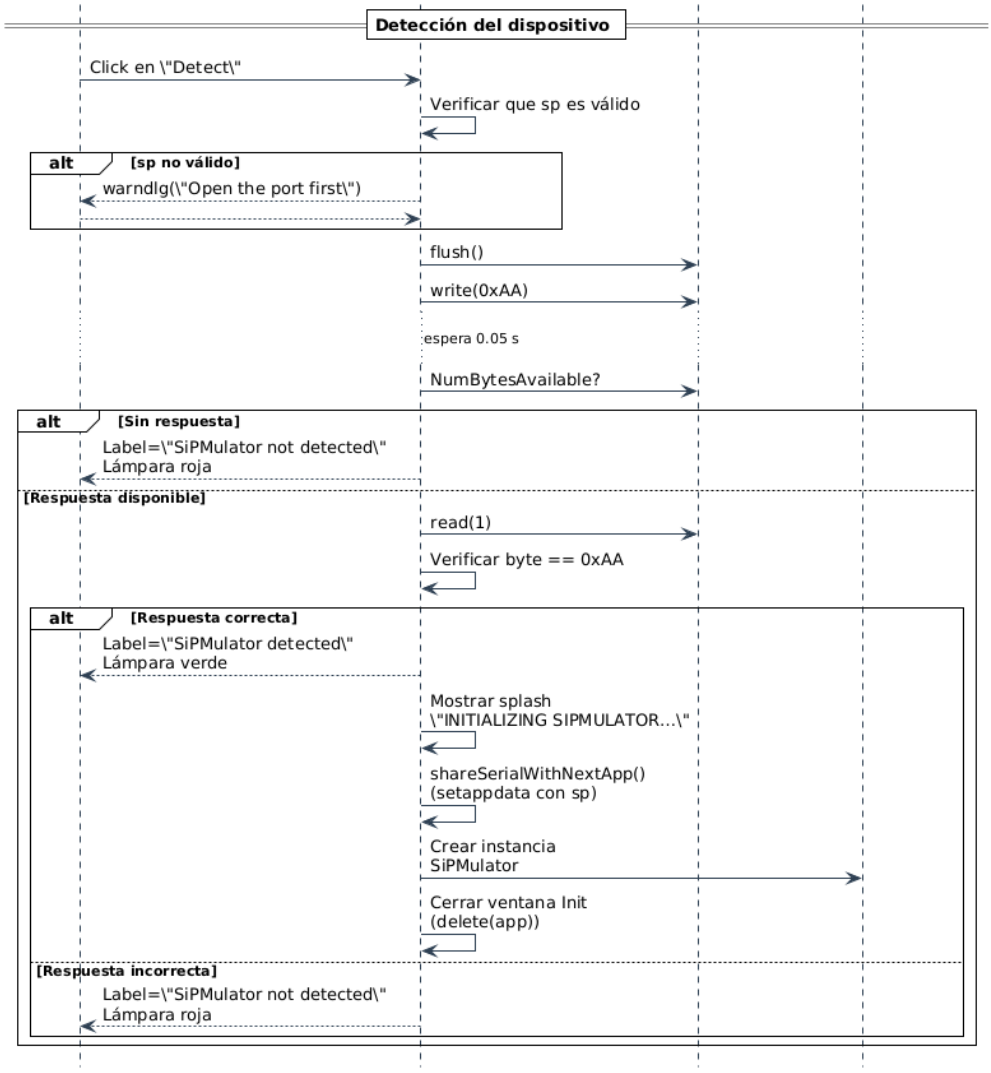
2.3.Modulos auxiliares:

- Módulos de procesamiento: sipm_pulse_to_32, plot_pulse_32_csv.
- Módulo de comunicación: SerialSession.

3. Flujo de inicialización

El proceso incluye: detección del puerto, apertura del puerto serial, envío del comando 0xAA, recepción de respuesta, ventana de inicialización y transferencia de la sesión serial a la aplicación principal, este flujo garantiza que SiPMulator esté conectado antes de lanzar la aplicación completa.



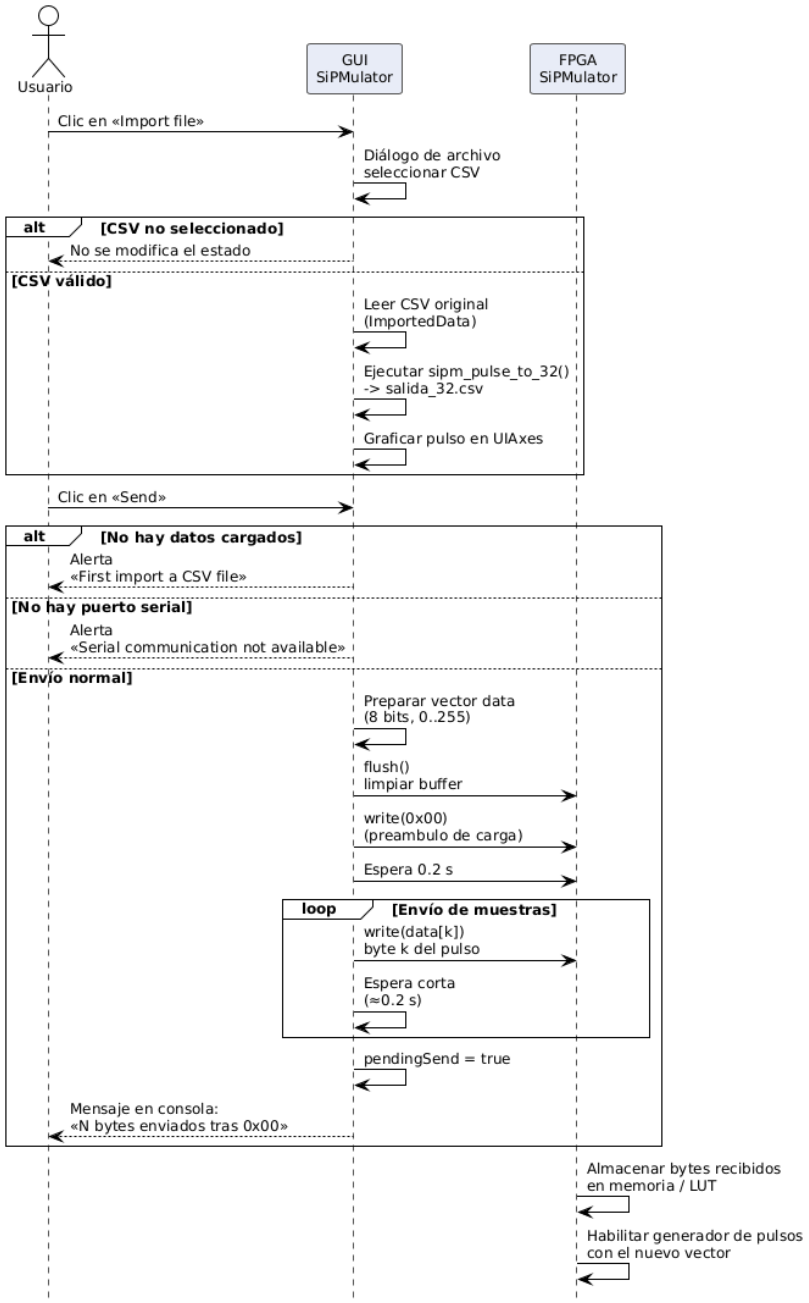


4. Comunicación UART con la FPGA

La comunicación opera bajo un protocolo simple de comandos:

- 0xAA: handshake.
- 0xFF: preámbulo antes de un cambio de modo.
- 0xBB: ruido.
- 0xCC: pulso con ruido.
- 0xDD: pulso periódico.
- 0x00: preámbulo para carga de pulsos de usuario.

La aplicación implementa validación, manejo de errores y reconexiones.



5. Procesamiento de pulsos

Basado en sipm_pulse_to_32, este módulo convierte un pulso original de ~12–15 muestras en un vector estándar de 32 muestras. El proceso incluye:

- Extracción de baseline (método auto/first/median).
- Ajuste exponencial de cola (modo fitexp).
- Interpolación pchip cuando el ajuste falla.
- Normalización al rango 8 bits.
- Forzado de baseline en muestra inicial y final.
- Escalamiento al pico deseado.

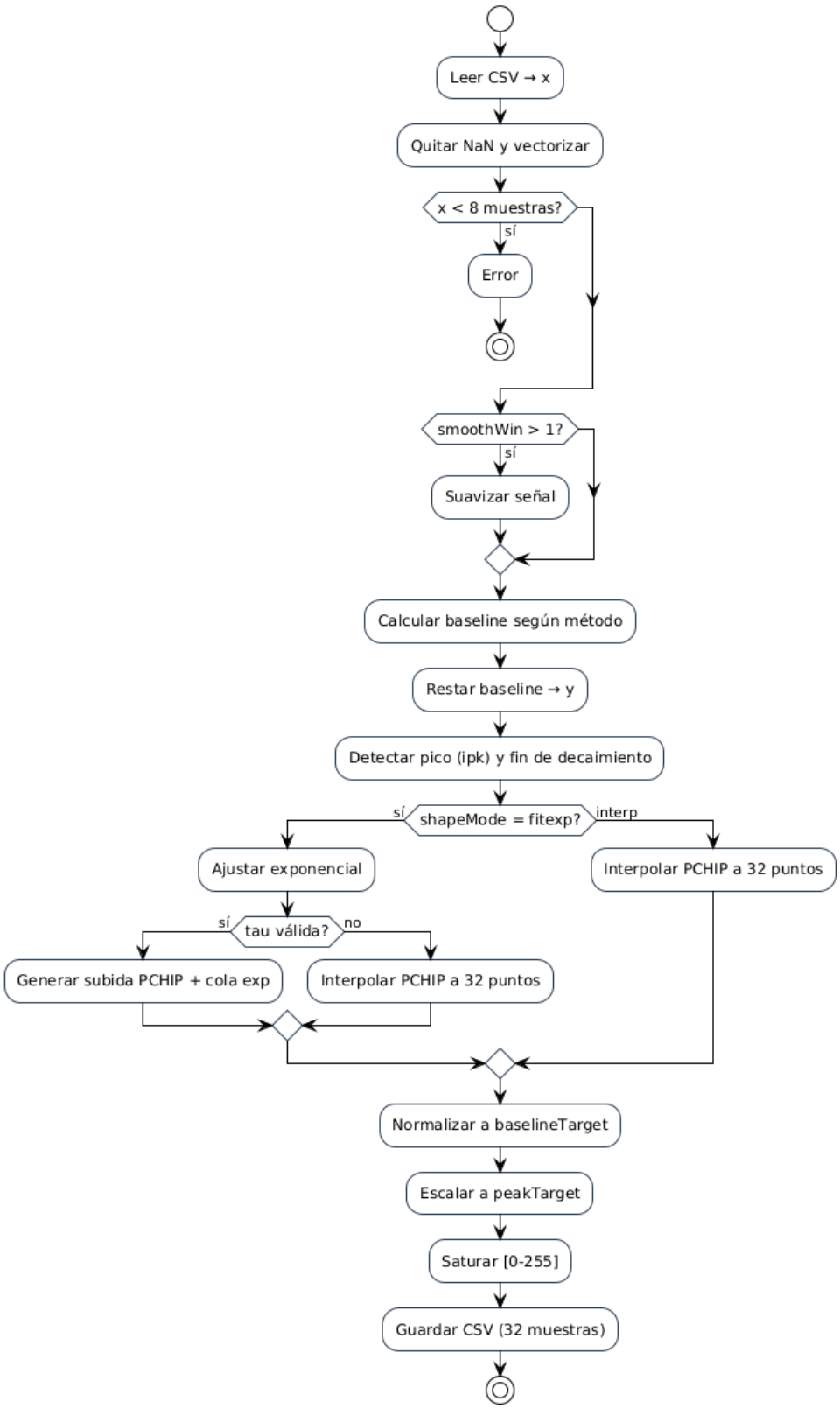
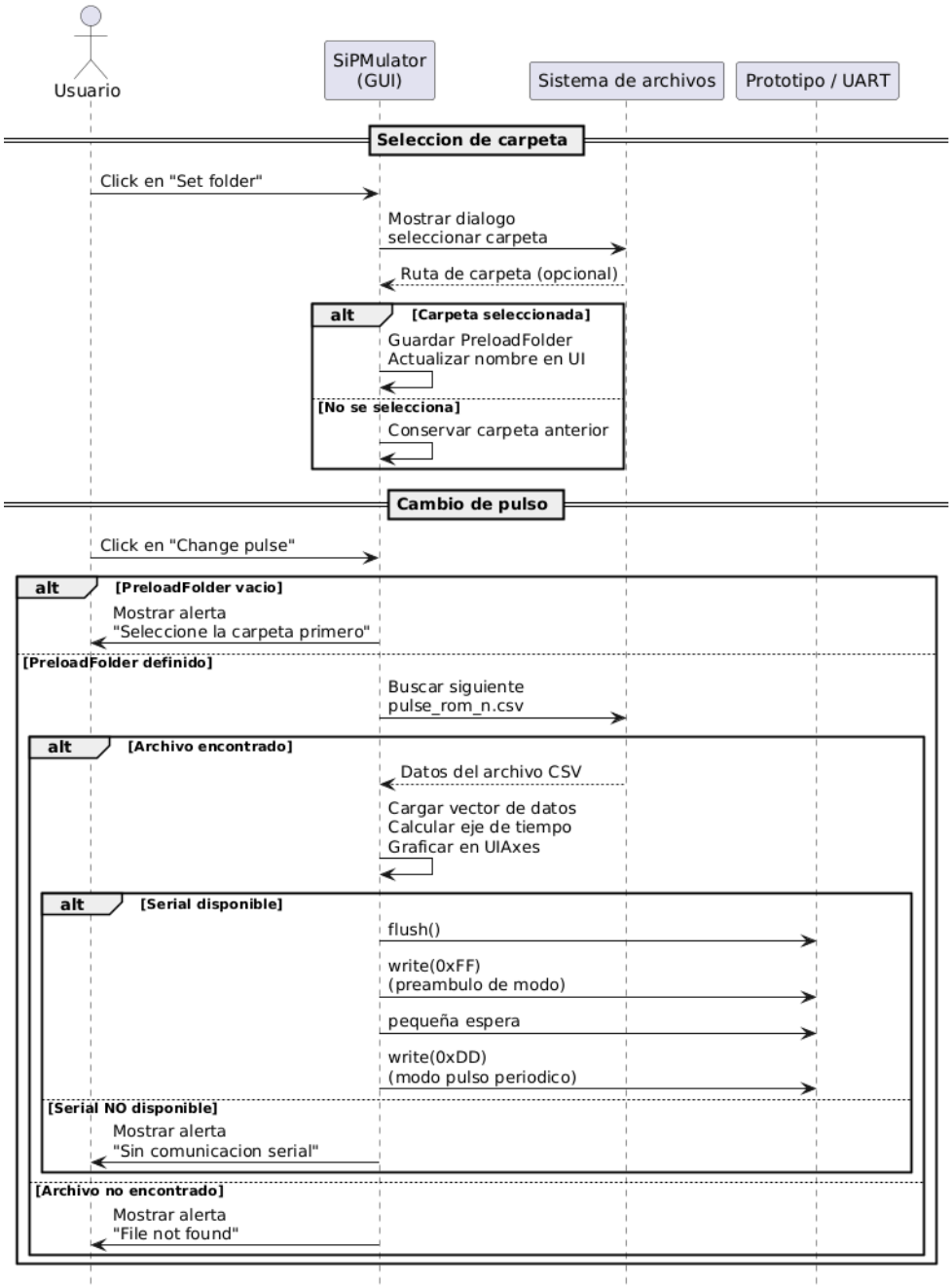

Leer CSV → x
Quitar NaN y vectorizar
x < 8 muestras?
sí
Error
smoothWin > 1?
sí
Suavizar señal
Calcular baseline según método
Restar baseline → y
Detectar pico (ipk) y fin de decaimiento
sí
shapeMode = fitexp?
interp
Ajustar exponencial
Interpolar PCHIP a 32 puntos
sí
tau válida?
no
Generar subida PCHIP + cola exp
Interpolar PCHIP a 32 puntos
Normalizar a baselineTarget
Escalar a peakTarget
Saturar [0-255]
Guardar CSV (32 muestras)

Diagrama matemático del ajuste exponencial

6. Gestión de pulsos pregrabados

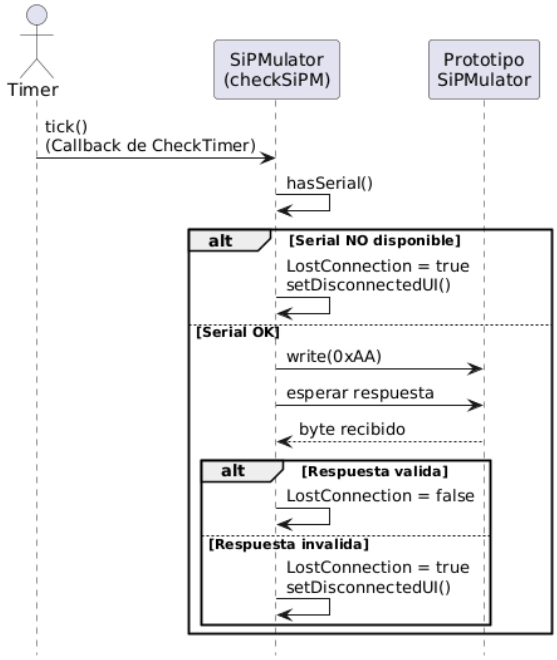
El sistema permite seleccionar una carpeta que contenga pulse_rom_n.csv, cada archivo se gráfica y puede enviarse automáticamente al prototipo junto con el modo correspondiente.



7. Control de estado y validación

El sistema implementa:

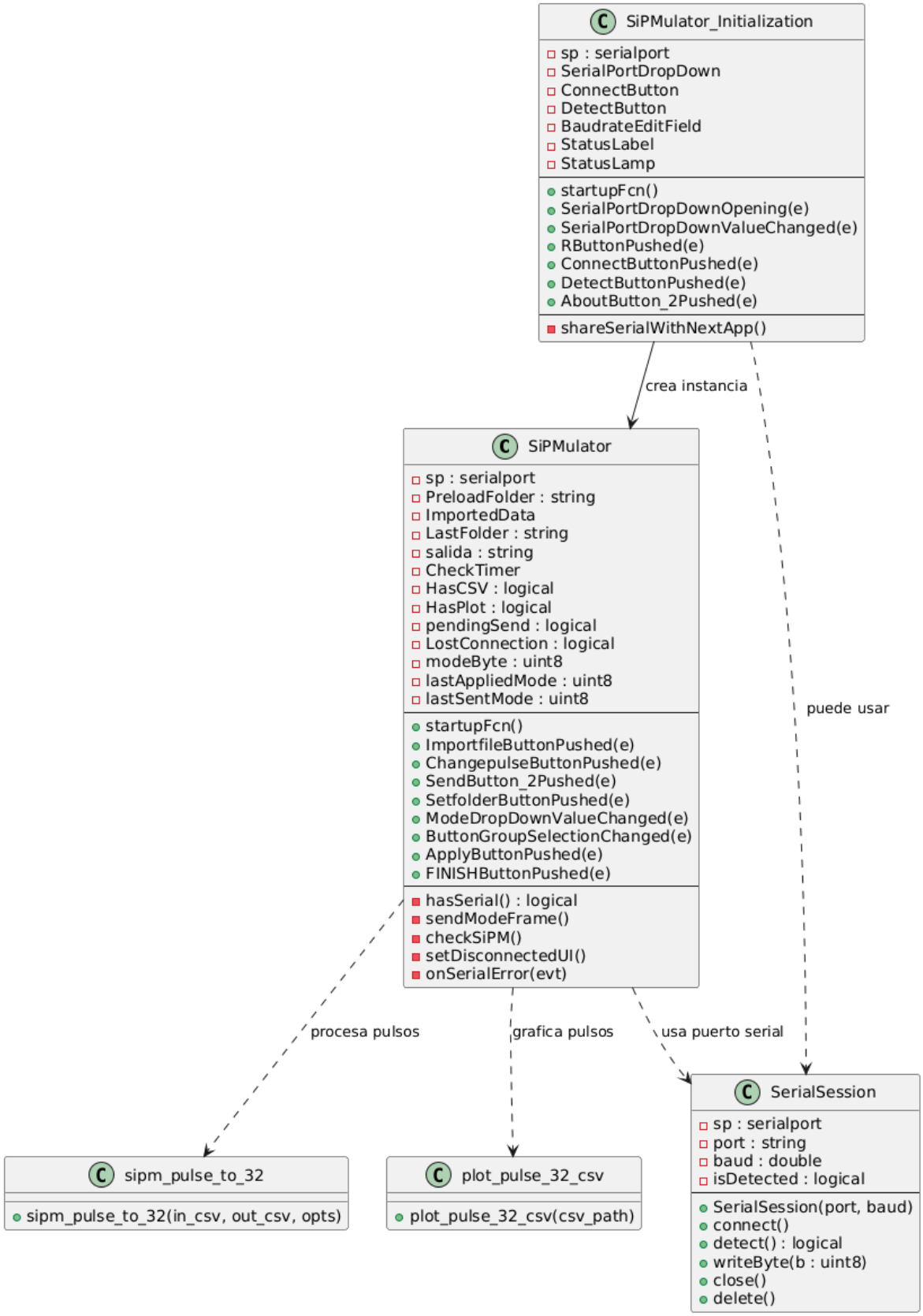
- Timer de verificación (1 Hz).
- Detección continua del estado del SiPMulator.
- Manejo de desconexión con apagado de UI.
- Alertas automáticas cuando se pierde la comunicación.



8. Diagrama de clases del sistema

Clases principales:

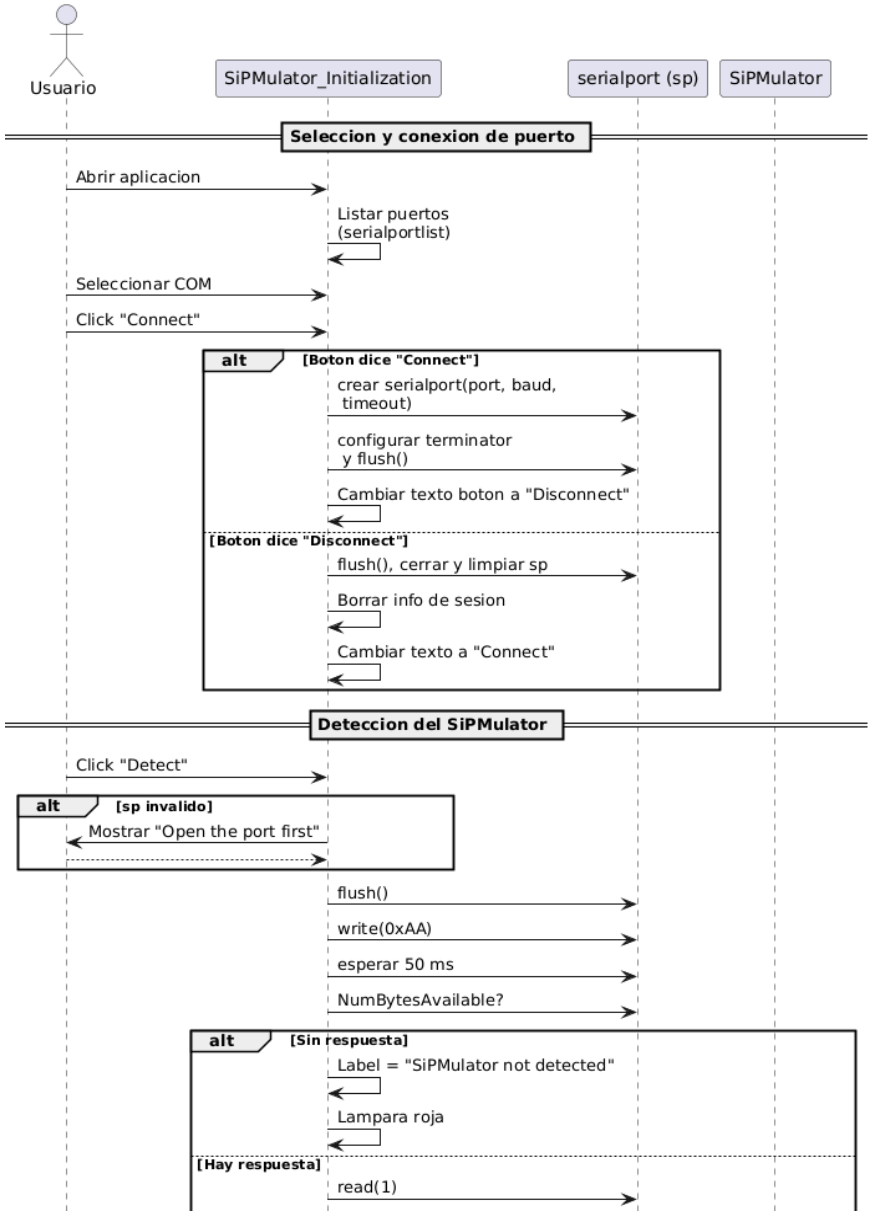
- SiPMulator: lógica central del control, envío de modos, carga de CSV, graficación y timer.
- SiPMulator_Initialization: establece la comunicación y lanza la aplicación principal.
- SerialSession: clase auxiliar para creación y manejo del puerto serial.

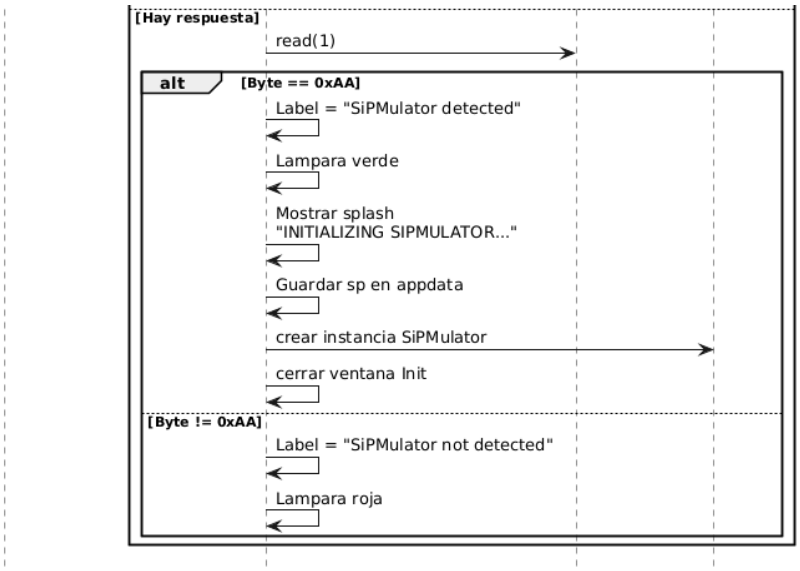

SiPMulator_Initialization
- sp : serialport
- SerialPortDropDown
- ConnectButton
- DetectButton
- BaudrateEditField
- StatusLabel
- StatusLamp
+ startupFcn()
+ SerialPortDropDownOpening(e)
+ SerialPortDropDownValueChanged(e)
+ RButtonPushed(e)
+ ConnectButtonPushed(e)
+ DetectButtonPushed(e)
+ AboutButton_2Pushed(e)
- shareSerialWithNextApp()
crea instancia
SiPMulator
- sp : serialport
- PreloadFolder : string
- ImportedData
- LastFolder : string
- salida : string
- CheckTimer
- HasCSV : logical
- HasPlot : logical
- pendingSend : logical
- LostConnection : logical
- modeByte : uint8
- lastAppliedMode : uint8
- lastSentMode : uint8
+ startupFcn()
+ ImportfileButtonPushed(e)
+ ChangepulseButtonPushed(e)
+ SendButton_2Pushed(e)
+ SetfolderButtonPushed(e)
+ ModeDropDownValueChanged(e)
+ ButtonGroupSelectionChanged(e)
+ ApplyButtonPushed(e)
+ FINISHButtonPushed(e)
- hasSerial() : logical
- sendModeFrame()
- checkSiPM()
- setDisconnectedUI()
- onSerialError(evt)
puede usar
procesa pulsos
grafica pulsos
usa puerto serial
sipm_pulse_to_32
+ sipm_pulse_to_32(in_csv, out_csv, opts)
plot_pulse_32_csv
+ plot_pulse_32_csv(csv_path)
SerialSession
- sp : serialport
- port : string
- baud : double
- isDetected : logical
+ SerialSession(port, baud)
+ connect()
+ detect() : logical
+ writeByte(b : uint8)
+ close()
+ delete()

9. Diagramas de secuencia lógica operativa del sistema

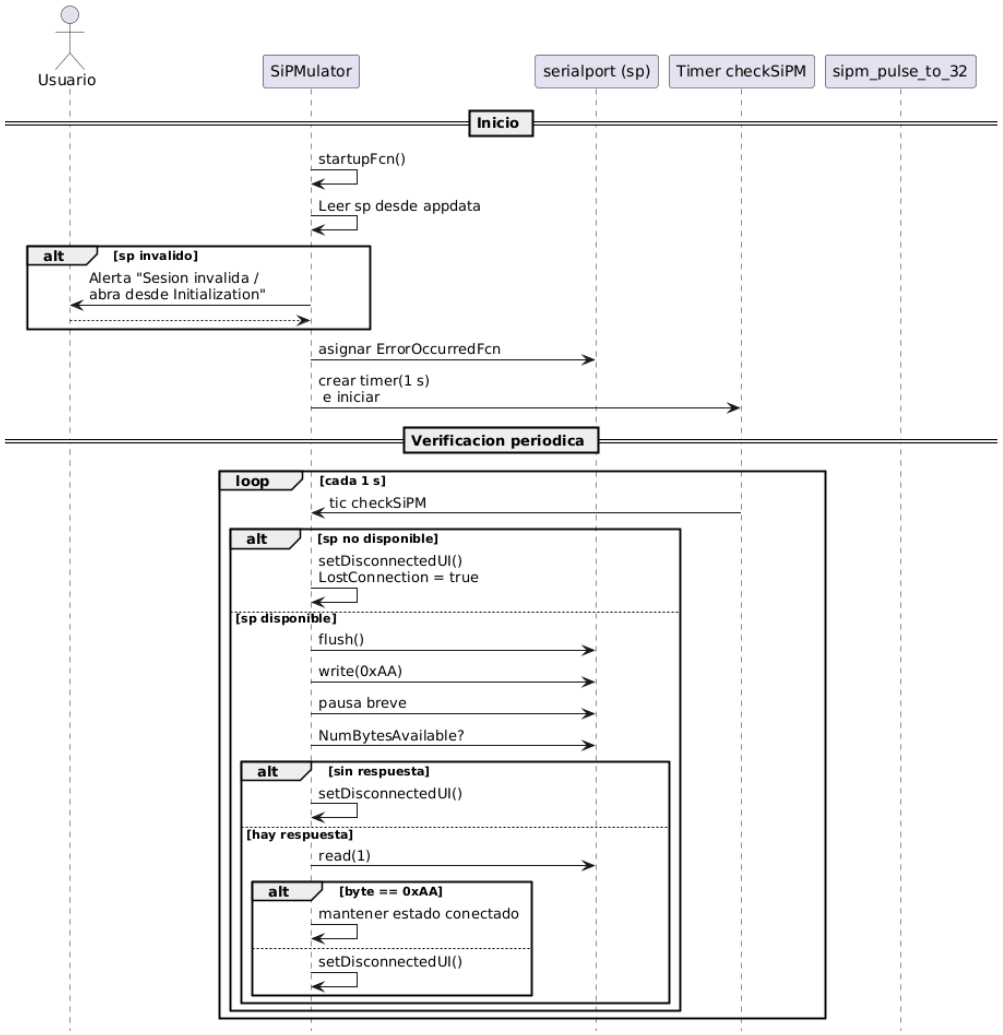
Los siguientes diagramas de secuencia describen la interacción entre el usuario, las interfaces gráficas y los módulos internos del SiPMulator, en ellos se detalla el orden exacto de llamadas, validaciones y mensajes intercambiados durante la conexión del puerto serial, la detección del dispositivo, la carga y envío de pulsos, el cambio de modos y la finalización de la sesión. Estos diagramas permiten visualizar con precisión el comportamiento dinámico del sistema y la comunicación entre sus componentes.

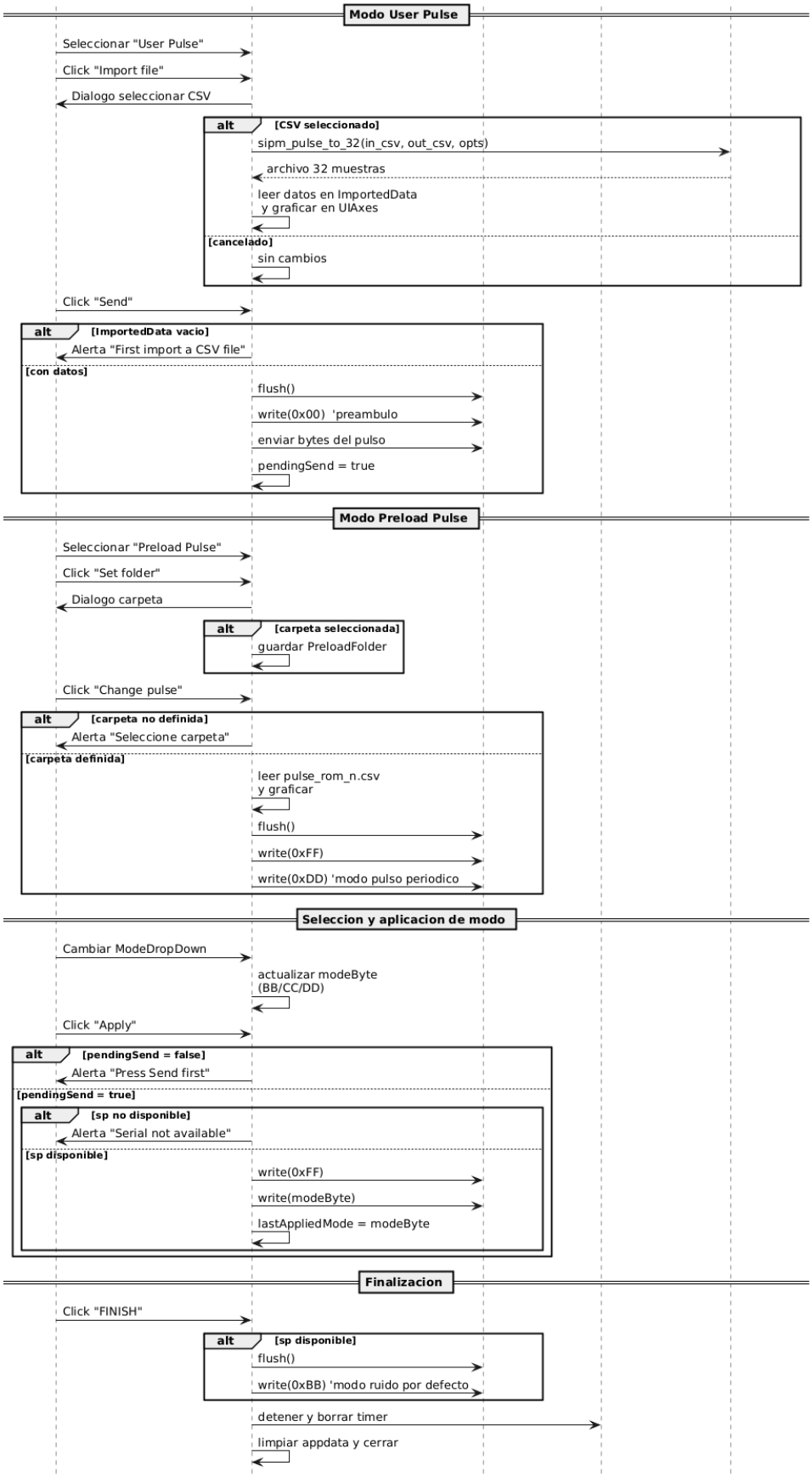
SiPMulator Initialization:




[Hay respuesta]
read(1)
alt
[Byte == 0xAA]
Label = "SiPMulator detected"
Lampara verde
Mostrar splash
"INITIALIZING SIPMULATOR..."
Guardar sp en appdata
crear instancia SiPMulator
cerrar ventana Init
[Byte != 0xAA]
Label = "SiPMulator not detected"
Lampara roja

SiPMulator


Usuario
SiPMulator
serialport (sp)
Timer checkSiPM
sipm_pulse_to_32
Inicio
startupFcn()
Leer sp desde appdata
alt
[sp invalido]
Alerta "Sesion invalida /
abra desde Initialization"
asignar ErrorOccurredFcn
crear timer(1 s)
e iniciar
Verificacion periodica
loop
[cada 1 s]
tic checkSiPM
alt
[sp no disponible]
setDisconnectedUI()
LostConnection = true
[sp disponible]
flush()
write(0xAA)
pausa breve
NumBytesAvailable?
alt
[sin respuesta]
setDisconnectedUI()
[hay respuesta]
read(1)
alt
[byte == 0xAA]
mantener estado conectado
setDisconnectedUI()


Modo User Pulse
Seleccionar "User Pulse"
Click "Import file"
Dialogo seleccionar CSV
alt [CSV seleccionado]
sipm_pulse_to_32(in_csv, out_csv, opts)
archivo 32 muestras
leer datos en ImportedData
y graficar en UIAxes
[cancelado]
sin cambios
Click "Send"
alt [ImportedData vacio]
Alerta "First import a CSV file"
[con datos]
flush()
write(0x00) 'preambulo
enviar bytes del pulso
pendingSend = true
Modo Preload Pulse
Seleccionar "Preload Pulse"
Click "Set folder"
Dialogo carpeta
alt [carpeta seleccionada]
guardar PreloadFolder
Click "Change pulse"
alt [carpeta no definida]
Alerta "Seleccione carpeta"
[carpeta definida]
leer pulse_rom_n.csv
y graficar
flush()
write(0xFF)
write(0xDD) 'modo pulso periodico
Seleccion y aplicacion de modo
Cambiar ModeDropDown
actualizar modeByte
(BB/CC/DD)
Click "Apply"
alt [pendingSend = false]
Alerta "Press Send first"
[pendingSend = true]
alt [sp no disponible]
Alerta "Serial not available"
[sp disponible]
write(0xFF)
write(modeByte)
lastAppliedMode = modeByte
Finalizacion
Click "FINISH"
alt [sp disponible]
flush()
write(0xBB) 'modo ruido por defecto
detener y borrar timer
limpiar appdata y cerrar

10. Gestión de Errores
El sistema SiPMulator maneja:

- Errores de puerto serial (puerto inexistente, desconectado, acceso denegado).
- Errores de carga de archivos (formatos inválidos, CSV vacío, tamaño incorrecto).
- Errores de comunicación UART (timeout, no respuesta).
- Errores de estado en la interfaz (UI bloqueada, reconexión requerida).

Todos los errores despliegan mensajes mediante uialert.

Créditos y referencias
Autora: Slyán Sophia Pinzón Miramón.
Director: Dr. Javier Fernando Castaño.
Asesor: Dr. Deywis Moreno.
Proyecto de grado: Desarrollo de un prototipo sistema generador de señales analógicas para pruebas experimentales de DAPHNE.
Semillero de investigación SITAD – Universidad Antonio Nariño, 2025.