Guide to Sending Tonic and Phasic Triggers from PsychoPy to a DSI-24 EEG System

January 13, 2025

1 1. Introduction

In many EEG experiments, it's essential to send **event markers** (also called *triggers*) from your behavioral task software (e.g., **PsychoPy**) to your EEG amplifier (e.g., **DSI-24**). These markers help you align the recorded EEG signals with specific events in your paradigm (e.g., stimulus onsets, responses, condition changes).

In this guide, you will learn: 1. The **difference** between **Tonic** (sustained) triggers and **Physical** (often called *Phasic*) triggers. 2. How to **open and manage** the serial connection to a **DSI-24** system. 3. **Example code** (in Python/PsychoPy) that demonstrates each trigger method in the context of a **Go/No-Go** experiment.

The explanations aim to be beginner-friendly while still maintaining the technical accuracy needed for proper EEG event marking.

---

2 2. Background on DSI-24 and Triggering

2.1 1.1 Hardware and Serial Communication

The **DSI-24** system is a wireless EEG headset that can receive numeric trigger values through a specialized interface (often via a **serial port** or other I/O device). These numeric codes are recorded in parallel with the EEG data, allowing you to later identify exactly when certain events occurred. ## 2.1 Physical Connection - You typically connect your computer (running PsychoPy) to the DSI system's data acquisition software (like **DSI-Streamer**) via a USB-to-serial interface or a dedicated trigger interface. When you connect the USB-to-serial interface to the DSI and also the Pscyhopy system you need to find the COM port associated with the USB-to-serial interface in the system using the Psychopy.

Finding Your COM Port
- **Windows**:
1. Open **Device Manager** (press Win+X → select "Device Manager").
2. Expand **"Ports (COM & LPT)"**.
3. Look for something like **"USB Serial Port (COMX)"**. The `X` in `COMX` is your COM port number.
- **Linux**:
1. Plug in your device.

2. Open a terminal and type `dmesg | grep tty` to see which device was assigned.
3. Common names are `/dev/ttyUSB0`, `/dev/ttyACM0`, etc.
- **macOS**:
1. Open the **Terminal**.
2. Type `ls /dev/tty.*` or `ls /dev/cu.*` to list serial devices.
3. Look for something referencing `usbserial` or similar.

2.2 2.2 Opening the Serial Port

Use Python's built-in `serial` library (or `pyserial`) in your PsychoPy script to open and configure the port. Example:

```
import serial

# Replace 'COM3' with the actual port name on your machine.
# For macOS or Linux, this may look like '/dev/ttyUSB0'
ser = serial.Serial('COM3')
ser.write(str.encode(chr(0)))  # Reset trigger line to 0 at the start
print("DSI-24 serial port opened. Trigger line set to 0.")
```

---

3 3. Tonic vs. Phasic Triggers: When to Use Which?

In EEG experiments, triggers mark key moments in your task so that recorded brain signals can be linked back to specific events or conditions. Two main styles are commonly used: **Tonic** (sustained) triggers and **Phasic** (brief) triggers. Choosing the right approach depends on how long your events last, how precisely you need to mark them, and what your subsequent analysis requires.

3.1 3.1 Tonic (Sustained) Triggers

3.1.1 Definition

A **tonic trigger** is set to a nonzero value at the start of an event or condition—such as a block of trials or a prolonged task state—and remains at that value until you explicitly reset it to zero.

3.1.2 When to Use Tonic Triggers

- **Long Blocks/States**
  If your experiment has extended intervals (e.g., a 2-minute "Task" vs. a 2-minute "Rest"), a tonic trigger continuously indicates which block is active.

- **Multi-Step Trials**
  In more complex or lengthy trials, you might keep a tonic trigger high and briefly add phasic pulses within that same period for sub-events.

- **Neurofeedback or Closed-Loop**
  Real-time systems often benefit from a constant marker to distinguish "active task" versus "baseline" phases.

3.1.3 Pros

- **Easy to Visualize**: You can see in the raw data exactly when a condition is active.
- **Simpler Setup for Long Durations**: Once you set the trigger, it stays high until you change it.

3.1.4 Cons

- **Must Remember to Reset**: Forgetting to reset could cause confusion in later blocks.
- **Less Precision**: Not ideal for capturing brief events that occur within the sustained period.

3.2 3.2 Phasic (Brief) Triggers

3.2.1 Definition

A **phasic trigger** is a short pulse (e.g., 10–50 ms) that is turned on at a specific moment—like stimulus onset or a button press—and then immediately turned back off.

3.2.2 When to Use Phasic Triggers

- **Discrete Stimulus Onsets**
  In many ERP (Event-Related Potential) studies (e.g., P300 paradigms), each stimulus is marked by a quick pulse for precise time-locking.
- **Rapid-Fire Events**
  If multiple events happen in quick succession, phasic pulses ensure each one is separately recorded without overlap.
- **Short Tasks**
  If a stimulus or response is short-lived (like a quick beep or flash), a short trigger pulse aligns with it neatly.

3.2.3 Pros

- **High Temporal Precision**: Excellent for analyzing reaction times or detecting ERP components.
- **Reduces Overlap**: The trigger resets to 0 quickly, so you won't accidentally obscure the next event.

3.2.4 Cons

- **No Long-Duration Indication**: You lose a continuous marker of an extended condition or block.
- **Hardware Sensitivity**: Some EEG amplifiers require a minimum pulse width (e.g., 10 ms) to reliably detect the trigger.

3.3 3.3 Choosing the Right Approach

- **Experiment Duration**
  - If your event or block lasts seconds to minutes and you want to visualize that entire period in the data, use **tonic**.
  - If you only need to mark precise, short-lived events, choose **phasic**.
- **Analysis Goals**
  - **ERP or Time-Locked** analyses (e.g., measuring latencies or reaction times) typically rely on **phasic** triggers.
  - **Block-Based** or **Neurofeedback** designs often benefit from **tonic** triggers to indicate when each condition starts and ends.
- **Possible Hybrid**
  Many experiments combine both: a tonic trigger to denote the overall block (e.g., "Condition A"), plus phasic triggers to mark each stimulus or response within that block.

3.4 3.4 Summary

- **Tonic = Sustained** marker (high throughout a block or long event).
- **Phasic = Brief Pulse** (ideal for short, discrete events).
- **Hybrid Approach**: Use tonic for broad phases and phasic for precise event timing.

Selecting the right style—or a combination—ensures your EEG recordings reflect exactly what's happening in your task. Clear triggers help you align neural data with behavioral events, making your subsequent analysis more accurate and meaningful.

---

4 4. Immediate vs. On-Flip Trigger Timing

Beyond selecting **tonic** or **phasic** triggers, you also need to decide **when** in the PsychoPy drawing cycle these triggers should be sent. Timing can be handled in two main ways:

1. **Immediate**: The trigger is sent the moment your code executes that line.
2. **On-Flip**: The trigger is scheduled to fire precisely at the **next screen refresh** (`win.flip()`).

Why does this matter? In EEG and psychophysics, you often want to **synchronize** your triggers with **visual events**. If your display updates at a certain frame, sending a trigger "immediately" may not perfectly align with that visual onset. Conversely, if the trigger is not tied to any visual event (e.g., it's only marking an internal logic change or a participant's key press), an immediate trigger might suffice.

4.1 4.1 Immediate Trigger Calls

When do they happen?
They occur exactly at the line in your code where you call the trigger function—there's no waiting for a screen refresh.

Use Case
- **Non-visual events** or purely logic-based changes. For instance, sending a trigger to mark the moment a key press is detected, or when your code transitions from one internal state to another.
- If the event you're marking doesn't need to coincide with a **stimulus flip**, an immediate call is straightforward and effective.

Example

```
# Immediately send a short (phasic) trigger to mark a key press
if 'space' in key_resp.keys:
    send_trigger_phasic(ser, 10, dur=0.01)
```

4.2 4.2 On-Flip Trigger Calls

When do they happen?
At the **next screen refresh**—the moment `win.flip()` executes in PsychoPy.

Use Case
- **Frame-locked** or **stimulus-locked** events: When you need a trigger to coincide precisely with a new stimulus appearing on screen. This is crucial in EEG studies requiring exact alignment with stimulus onset.

Example

```
def turn_trigger_on():
    ser.write(str.encode(chr(20)))  # e.g., trigger value 20
    print("Trigger 20 sent at screen flip")
win.callOnFlip(turn_trigger_on)
win.flip()
core.wait(0.05)  # hold for 50 ms if phasic
ser.write(str.encode(chr(0)))  # reset line back to 0
```

Here, the trigger sets to value 20 exactly when the screen buffers swap, meaning the participant sees the new stimulus and the EEG marker changes in the same instant.

4.3 4.3 Choosing the Optimal Timing Approach

1. **Immediate**:

- Simpler for marking background logic, user responses, or anything not explicitly tied to the next screen refresh.
- Slightly less precise for visual onset, but sufficient if you don't need strict frame-locking.

2. **On-Flip**:

- Essential if your stimulus onset marker must match the exact frame the participant sees the stimulus.
- The standard in tasks where accurate time-locking is mandatory (e.g., ERP paradigms).

In practice, many experiments use both:

- On-Flip triggers to ensure stimulus presentation is accurately time-locked in the EEG data.
- Immediate triggers to mark events like button presses or internal state changes that do not depend on a specific screen refresh cycle.

---

5 5. Implementation: Tonic & Phasic Trigger Functions

Having seen the differences between **Tonic** vs. **Phasic** triggers (Section 3) and **Immediate** vs. **On-Flip** timing (Section 4), you're now ready to integrate these concepts into your own PsychoPy experiments. Below, you'll find **ready-to-use** Python functions that illustrate how to send triggers via a serial port in both a phasic (brief) and tonic (sustained) manner, as well as how to schedule them **immediately** or **at the next screen flip**.

5.1 5.1 Overview of Provided Functions

1. **Phasic (Brief) Triggers**
   - `send_trigger_phasic(ser, data, dur=0.01)`
     - Sends a short, immediate pulse (e.g., 10–50 ms).
   - `send_trigger_onflip_phasic(win, ser, trigger_value, dur=0.05)`
     - Sends a short pulse **on the next screen flip**, then resets it after `dur` seconds.
2. **Tonic (Sustained) Triggers**
   - `send_trigger_tonic(ser, trigger_value)`
     - Immediately sets the trigger line to a nonzero value and **holds** it until you reset it.
   - `reset_trigger_tonic(ser)`
     - Resets a tonic trigger back to 0.
   - `send_trigger_onflip_tonic(win, ser, trigger_value)`
     - Schedules a tonic trigger to begin **on the next screen flip**, and it stays "on" until you manually reset it.

These functions give you the **fine-grained control** to either: - **Trigger Immediately** (for non-visual events or logic-based signals), or
- **Trigger On-Flip** (for precise alignment with screen refresh and stimulus onset).

Each code snippet below includes a docstring describing how it works, the arguments it takes, and typical usage notes.

5.2 5.2 Code Snippets

"'python

from psychopy import core

def send_trigger_phasic(ser, data, dur=0.01): " " " Send a brief (phasic) trigger immediately.

```
- 'ser': An open serial.Serial object for communication.
- 'data': Integer representing the trigger code to send.
- 'dur': How long (in seconds) to keep the trigger active before resetting to 0.

Usage Example:
    send_trigger_phasic(ser, 10, dur=0.02)
```

```
    # Instantly sets the line to 10, holds for ~20 ms, then returns to 0.
"""
# Record the start time to manage the pulse length accurately
start = core.getTime()

# Send the trigger value
ser.write(str.encode(chr(data)))

# Calculate remaining time to hold the trigger
elapsed = core.getTime() - start
remaining = dur - elapsed

# If the time to send the trigger was shorter than 'dur', wait the difference
if remaining > 0:
    core.wait(remaining)

# Reset the trigger to 0 after the pulse
ser.write(str.encode(chr(0)))
```

def send_trigger_onflip_phasic(win, ser, trigger_value, dur=0.05): “ “ ” Schedule a brief (phasic) trigger to fire at the next screen flip, then reset.

```
- 'win': A psychopy.visual.Window object where you call 'win.flip()'.
- 'ser': An open serial.Serial object for communication.
- 'trigger_value': Integer code to send.
- 'dur': Duration (in seconds) to keep the trigger active.

After 'win.flip()', we hold the trigger line high for 'dur' seconds,
then automatically reset it to 0.

Usage Example:
    send_trigger_onflip_phasic(win, ser, trigger_value=20, dur=0.05)
    # The line will go to 20 exactly when win.flip() is executed,
    # remain high for 50 ms, then reset to 0.
"""

def turn_trigger_on():
    ser.write(str.encode(chr(trigger_value)))
    print(f"[PHASIC-ON-FLIP] Trigger {trigger_value} sent at screen flip")

# 1) Queue the "turn on" function for the next flip
win.callOnFlip(turn_trigger_on)

# 2) Execute the next flip
win.flip()

# 3) Hold the trigger for 'dur' seconds
core.wait(dur)
```

```
# 4) Reset the trigger line to 0
ser.write(str.encode(chr(0)))
print("[PHASIC-ON-FLIP] Trigger reset to 0")
```

def send_trigger_tonic(ser, trigger_value): “ “ ” Set a ‘tonic’ (sustained) trigger immediately.

```
- 'ser': An open serial.Serial object.
- 'trigger_value': Integer code to send.

The trigger will remain at 'trigger_value' until you manually reset it.

Usage Example:
    send_trigger_tonic(ser, 5)
    # The line will stay at 5 indefinitely, or until reset_trigger_tonic() is called.
"""
ser.write(str.encode(chr(trigger_value)))
print(f"[TONIC] Trigger set to {trigger_value} (must reset later).")
```

def reset_trigger_tonic(ser): “ “ ” Reset a tonic trigger back to 0.

```
- 'ser': An open serial.Serial object.

Usage Example:
    reset_trigger_tonic(ser)
    # Immediately sets the line back to 0.
"""
ser.write(str.encode(chr(0)))
print("[TONIC] Trigger reset to 0.")
```

def send_trigger_onflip_tonic(win, ser, trigger_value): “ “ ” Schedule a ‘tonic’ trigger to begin exactly on the next screen flip.

```
- 'win': A psychopy.visual.Window object.
- 'ser': An open serial.Serial object.
- 'trigger_value': Integer code to send.

This line remains active until you call reset_trigger_tonic().
Ideal for block-based marking where exact alignment with stimulus onset is critical.

Usage Example:
    send_trigger_onflip_tonic(win, ser, 9)
    win.flip()
    # The line goes to 9 on that flip and stays high until reset_trigger_tonic(ser).
"""

def turn_trigger_on():
    ser.write(str.encode(chr(trigger_value)))
    print(f"[TONIC-ON-FLIP] Trigger {trigger_value} set at screen flip (must reset later).")

# Schedule the trigger for the next screen refresh
win.callOnFlip(turn_trigger_on)
```

```
# Actually flip to trigger
win.flip()
```

5.3 5.3 Practical Tips for Using These Functions

1. **Initialize Your Serial Port First**

   Before you call any trigger functions, open and configure the serial port. For example:

   ```
   import serial
   ser = serial.Serial('COM3')  # Replace 'COM3' with the correct port
   ser.write(str.encode(chr(0)))  # Ensure the line starts at 0
   ```

   This step ensures that triggers can be sent correctly and that the line is initially set to zero.

2. **Define Your Trigger Codes**

   It's helpful to keep a clear mapping of experimental events to numeric codes: `python GO_STIM_ONSET = 10 NO_GO_STIM_ONSET = 11 RESPONSE_CORRECT = 20 RESPONSE_INCORRECT = 21` This makes your code more readable and maintain consistency throughout your experiment.

3. **Combine Tonic & Phasic Where Needed**

   - Tonic triggers are ideal for marking entire blocks or extended phases. Phasic triggers are perfect for brief events (e.g., stimulus onset, button press) within those blocks. By using both, you maintain a clear separation of long-running states and short, discrete events.
   - Phasic triggers are perfect for brief events (e.g., stimulus onset, button press) within those blocks.

   By using both, you maintain a clear separation of long-running states and short, discrete events.

4. **Always Reset Tonic Triggers**

   If you call `send_trigger_tonic(ser, value)`, remember to call `reset_trigger_tonic(ser)` once that block or condition ends. Otherwise, your EEG recording may remain at the same trigger value indefinitely, making later events harder to interpret.

5. **Check Timing Requirements**

   Minimum Pulse Width: Confirm your EEG system's specs; many require at least

---

6 6. Example Task: Go/No-Go on Pavlovia

This section demonstrates how to incorporate triggers (both **phasic** and **tonic**) into a **Go/No-Go** experiment. The same logic applies to any other task; simply adapt the code for your events and conditions. We've also uploaded an example task to Pavlonia for the Phasic and Tonic approach so you can see the implementation in action.

6.1 6.1 High-Level Task Flow

A typical **Go/No-Go** experiment might follow these steps:

1. **Task Start**
   - Optionally send a "task start" trigger, either as a brief (phasic) pulse or a tonic signal.
2. **Trial Begins**
   - Display either a **Go** or **No-Go** stimulus.
   - Immediately mark the stimulus onset with a phasic or tonic trigger.
3. **Participant Responds** (or not)
   - Collect keypresses.
   - Determine correctness (Go correct = pressed, No-Go correct = no press).
   - Send a trigger to mark correct or incorrect responses.
4. **Trial Ends**
   - If using **tonic** triggers, reset the line to 0 here (so it doesn't carry into the next trial).
5. **Repeat** for the desired number of trials.
6. **Task End**
   - Send a "task end" trigger.
   - Reset the line to 0, and close the serial port.

6.2 6.2 Adding Triggers in PsychoPy's Builder

We'll assume you have a Builder setup with: - A routine named **"trial"** for each Go/No-Go stimulus presentation. - A **keyboard component** named `key_resp` for participant responses. - A **Code Component** in that routine (plus potentially separate routines for "start_trigger" and "end_trigger").

6.2.1 6.2.1 "Begin Experiment" Tab

In your Code Component's **"Begin Experiment"** section in the `Welcome` routine, define the list of triggers and your functions as explained in Section 5.2, based on the type of trigger you are using (Phasic or tonic).

- Phasic

[ ]:
```
import serial
from psychopy import core, visual, event

# Initialize the serial port for the Neurospec MMBT-S -> DSI
Ser = serial.Serial('COM3')
```

```
# Immediately set trigger line to 0 at the experiment start
Ser.write(str.encode(chr(0)))
print("Serial port opened. Trigger line reset to 0.")

# Define trigger codes (integers for each event)
TASK_START   = 10
GO_ONSET     = 1
NOGO_ONSET   = 2
GO_CORRECT   = 3
NOGO_CORRECT = 4
INCORRECT    = 5
TASK_END     = 99

# Define your phasic/tonic trigger functions
def send_trigger_phasic(ser, data, dur=0.01):

    start = core.getTime()
    ser.write(str.encode(chr(data)))
    elapsed = core.getTime() - start
    remaining = dur - elapsed
    if remaining > 0:
        core.wait(remaining)
    ser.write(str.encode(chr(0)))


def send_trigger_onflip_phasic(win, ser, trigger_value, dur=0.05):

    def turn_trigger_on():
        ser.write(str.encode(chr(trigger_value)))
        print(f"Trigger {trigger_value} sent")
    win.callOnFlip(turn_trigger_on)
    win.flip()
    core.wait(dur)
    ser.write(str.encode(chr(0)))
    print("Trigger reset to 0")
```

- Tonic

[ ]:
```
import time
import serial
Ser=serial.Serial('COM3')
from psychopy import core

def send_trigger_tonic(ser, trigger_value):
    ser.write(str.encode(chr(trigger_value)))
    core.wait(0.05)
    print(f"[TONIC] Trigger set to {trigger_value} (must reset later).")
```

```
def reset_trigger_tonic(ser):
    ser.write(str.encode(chr(0)))
    print("[TONIC] Trigger reset to 0.")

def send_trigger_onflip_tonic(win, ser, trigger_value):
    def turn_trigger_on():
        ser.write(str.encode(chr(trigger_value)))
        print(f"[TONIC] Trigger {trigger_value} sent on next flip (must reset␣
 ↪later).")

    # Schedule the trigger for the next screen refresh
    win.callOnFlip(turn_trigger_on)

    # Perform the flip to execute the scheduled command
    win.flip()

def send_trigger_phasic(ser, data, dur=0.01):
    start = core.getTime()
    ser.write(str.encode(chr(data)))
    elapsed = core.getTime() - start
    remaining = dur - elapsed
    # Avoid negative waiting if sending took longer than dur
    if remaining > 0:
        core.wait(remaining)
    # Reset the trigger to 0
    ser.write(str.encode(chr(0)))
```

6.2.2 6.2.2 "Begin Routine" Tab (Stimulus Onset)

In the **"Begin Routine"** tab of your `trial` routine's Code Component, you can decide how to mark stimulus onset. For instance, if your trial dictionary is named `thisTrial` and has a key `'this_image'`:

- Phasic

[ ]:
```
# Example: phasic approach for exact onset
if thisTrial['this_image'] == 'go.png':
    send_trigger_phasic(Ser, GO_ONSET, dur=0.01)
elif thisTrial['this_image'] == 'nogo.png':
    send_trigger_phasic(Ser, NOGO_ONSET, dur=0.01)
```

- Tonic

[ ]:
```
# Decide which trigger to send depending on the image

if thisTrial['this_image'] == 'go.png':
    combined_trigger = TASK_TONIC_TRIGGER | GO_ONSET_TONIC_TRIGGER
```

```
    send_trigger_onflip_tonic(win,Ser, combined_trigger)
    print("GO_ONSET_TONIC_TRIGGER activated.")

elif thisTrial['this_image'] == 'nogo.png':
    combined_trigger = TASK_TONIC_TRIGGER | NOGO_ONSET_TONIC_TRIGGER
    send_trigger_onflip_tonic(win,Ser, combined_trigger)
    print("NOGO_ONSET_TONIC_TRIGGER activated.")
```

6.2.3 6.2.3 "End Routine" Tab (Response Evaluation)

After the participant responds (or doesn't), you can mark correct vs. incorrect outcomes. To implement this in the **"End Routine"** tab of the `trial` routine you can use the following code:

- Phasic

[ ]:
```
# Grab whatever keys were pressed this trial
resp_keys = key_resp.keys if key_resp.keys else []
print(resp_keys)

# Determine correctness based on image and response
if thisTrial['this_image'] == 'go.png':
    # For a Go trial, correct if the participant pressed 'space'
    if 'space' in resp_keys:
        send_trigger(Ser,GO_CORRECT_TRIGGER)
        print(f"Correct Go response, trigger {GO_CORRECT_TRIGGER} sent")
    else:
        send_trigger(Ser,INCORRECT_TRIGGER)
        print(f"Incorrect Go response, trigger {INCORRECT_TRIGGER} sent")

elif thisTrial['this_image'] == 'nogo.png':
    # For a NoGo trial, correct if NO key was pressed
    if len(resp_keys) == 0:
        send_trigger(Ser,NOGO_CORRECT_TRIGGER)
        print(f"Correct NoGo response, trigger {NOGO_CORRECT_TRIGGER} sent")
    else:
        send_trigger(Ser,INCORRECT_TRIGGER)
        print(f"Incorrect NoGo response, trigger {INCORRECT_TRIGGER} sent")

else:
    # In case there's some unexpected image name
    print("Warning: Unknown image type, no trigger sent.")
```

- Tonic

[ ]:
```
resp_keys = key_resp.keys if key_resp.keys else []
print(resp_keys)
# Determine correctness based on image and response
if thisTrial['this_image'] == 'go.png':
```

```
    # For a Go trial, correct if the participant pressed 'space'
    if 'space' in resp_keys:
        combined_trigger = TASK_TONIC_TRIGGER | CORRECT_RESP_TONIC_TRIGGER
        send_trigger_tonic(Ser, combined_trigger)
        print("CORRECT_RESP_TONIC_TRIGGER activated for Go trial.")
    else:
        combined_trigger = TASK_TONIC_TRIGGER | INCORRECT_RESP_TONIC_TRIGGER
        send_trigger_tonic(Ser, combined_trigger)
        print("INCORRECT_RESP_TONIC_TRIGGER activated for Go trial.")

elif thisTrial['this_image'] == 'nogo.png':
    # For a NoGo trial, correct if NO key was pressed
    if len(resp_keys) == 0:
        combined_trigger = TASK_TONIC_TRIGGER | CORRECT_RESP_TONIC_TRIGGER
        send_trigger_tonic(Ser, combined_trigger)
        print("CORRECT_RESP_TONIC_TRIGGER activated for No-Go trial.")
    else:
        combined_trigger = TASK_TONIC_TRIGGER | INCORRECT_RESP_TONIC_TRIGGER
        send_trigger_tonic(Ser, combined_trigger)
        print("INCORRECT_RESP_TONIC_TRIGGER activated for No-Go trial.")
# After sending the response trigger, reset to task-level tonic trigger
send_trigger_tonic(Ser, TASK_TONIC_TRIGGER)
print("Returned to TASK_TONIC_TRIGGER state.")
```