Museum Integration API / Interface

Raspberry Pi + IMX219 High-Speed Camera -> Polymerization Exhibit

Table of Contents

0) Purpose & Scope

This document defines a repeatable integration interface ("API") so a museum technologist can connect a Raspberry Pi + IMX219 camera to the existing polymerization exhibit and get a reliable, high-fps capture -> slow-motion playback loop. It specifies hardware connectors, electrical levels, timing, software commands, configuration, expected I/O artifacts, and validation tests. It assumes an existing droplet/UV prototype with Arduino-based control.

It is an **integration contract** across electronics, GPIO, serial commands, and command-line tools.

---

1) System Overview (Contract Boundaries)

Actors

- **Exhibit Controller (Arduino Nano)**: runs droplet, sensor, UV logic; emits a *trigger pulse* when a droplet enters the filmed region.
- **Capture Host (Raspberry Pi 3)**: runs modified `raspiraw`, listens for trigger on GPIO17, captures ~100 ms at ~950–1000 fps, saves raw frames + timestamps, then renders slow-motion.
- **Camera (Sony IMX219)**: rotated 90° to align rolling shutter with vertical droplet path; uses small vertical ROI to hit ~1 kfps.

Key Interfaces

- **Electrical trigger**: Arduino A3 -> (voltage divider) -> Pi GPIO17 (rising edge).
- **Serial control** (optional, for debug/ops): Pi/Computer ↔ Arduino over USB CDC; simple text commands (e.g., `go`, `stop`).
- **File outputs**: frames & timestamps in RAM-disk (`/dev/shm`), then playback via OpenCV scripts.

---

2) Hardware Integration API

2.1 Connectors & Pin Map

- **SENSE4 header on exhibit shield** (chosen to avoid collisions with existing wiring):
  - **Pin 1:** Vin (*not used)*.
  - **Pin 2: Arduino A3** (digital output *trigger*).
  - **Pin 3: GND** (must tie Arduino and Pi grounds together).

2.2 Electrical Levels

- **Arduino -> Pi trigger level shift:** passive divider 5 V -> ~3.2 V.
  - Example divider: **2 kΩ (top)** to Arduino A3, **1 kΩ (bottom)** to GND; midpoint to Pi GPIO17.
  - Result ≈ 5 V × (1 kΩ / (2 kΩ + 1 kΩ)) ≈ 3.3 V (safe for Pi input).

2.3 Raspberry Pi GPIO Contract

- **Trigger input pin:** GPIO17
- **Edge: Rising** edge starts capture.
- **Pulse width requirement:** ≥ 5 ms (debounced by firmware; longer is fine).
  If in doubt: 20–50 ms pulse width is robust.

2.4 Camera Mounting & Optics

- **Sensor rotation: 90°** so the vertical ROI tracks droplet fall.
- **ROI height presets:** 64 / 128 / 192 rows (trade coverage vs. fps). Default: **64 rows** for ~950–1000 fps.
- **Focus:** set with a hanging droplet under exhibit lighting; confirm edges are crisp within ROI.

---

3) Timing API

3.1 Event Sequence (Nominal)

```
[Droplet detected] -> Arduino emits A3 HIGH -> Pi sees rising edge on GPIO17 ->
raspiraw starts capture (target 100 ms @ ~1000 fps) -> frames buffered to RAM ->
Pi writes timestamps.csv and RAW frames -> playback script builds slow-motion -> HDMI
display.
```

3.2 Capture Window Policies

- **Baseline:** pre-armed, fixed **100 ms** capture.
- **Trimmed options (for faster turnaround):**
  - **P10/P90 policy:** pre-delay ~28 ms, record ~60 ms (edge miss risk bounded by chosen percentiles).
  - **Mild trim:** pre-delay ~10 ms, record ~80 ms.
- **Fallback:** if no frames arrive within 250 ms after trigger, abort and log an error.

---

4) Software/API Surface on Raspberry Pi

4.1 Required Software

- Modified `raspiraw` (IMX219-only build with dynamic ROI & ring buffer).
- Python 3 + OpenCV + NumPy (playback & utilities).
- `dcraw` or equivalent for RAW→TIFF (where applicable).

4.2 Command-Line Capture API (contract)

CLI: `raspiraw [options...]`

Required options:
`-md 7` -> IMX219 mode (compatible with small vertical ROI)
`-t <ms>` -> capture duration (e.g., `100`)
`--fps <fps>` -> requested frame rate (e.g., `1000`)
`-h <rows>` / `-w <cols>` -> ROI rows/cols (e.g., `-h 64 -w 640`)
`-o /dev/shm/out.%06d.raw` -> RAW frame pattern in RAM

`-ts /dev/shm/tstamps.csv` -> timestamp log
`-g <gain>` -> sensor analog gain (e.g., 1–8)
`-eus <µs>` -> exposure in microseconds (e.g., 750)
`--regs "0171,01;0170,01"` -> example register tweaks used in testing

Canonical example:

```
./raspiraw -md 7 -t 100 -ts /dev/shm/tstamps.csv -hd0 /dev/shm/hd0.32k \
  -h 64 -w 640 --fps 1000 -sr 1 --regs "0171,01;0170,01" -g 1 -eus 750 \
  -o /dev/shm/out.%06d.raw
```

4.3 GPIO Trigger Daemon (service)

- A small Python service arms capture and waits on GPIO17 rising edges.
- On trigger: spawn the `raspiraw` command with configured ROI/exposure, then enqueue **post-processing**.
- On completion: notify the **playback** process.

4.4 Playback API

- **Debug:** `player_tiff_keyboard.py` / `player_tiff_keyboard_interp.py` (interactive).
- **Exhibit mode:** `player_pin.py` (or a kiosk script) to autoplay most recent capture on HDMI.
- **Interpolation factor (optional):** ×4 for smoother slow-motion on public display.

4.5 Output Artifacts (contract)

- `/dev/shm/tstamps.csv`: frame index + µs timestamp per frame.
- `/dev/shm/out.XXXXXX.raw`: contiguous RAW frames.

---

5) Arduino-Side Interface

5.1 Firmware Behavior (integration contract)

- On droplet detect: assert **A3 HIGH** once for a single event; de-assert after UV completes.
- Avoid multiple pulses per droplet; minimum inter-event spacing: **≥ 1 s**.

5.2 Serial Command Set (for ops & debug)

```
help                # list commands
status              # print key parameters
set drnum <N>       # set number of drops in a run (if used)
set drprt <P>       # set droplet period (ms) or profile param
go                  # start a programmatic run (will emit A3 when appropriate)
stop                # stop run and de-energize actuators
```

---

6) Configuration Profiles (Examples)

6.1 High-FPS, Narrow View (default)

```
ROI: 64 rows, 640 cols
FPS: 1000
Exposure: 750 µs
```

```
Gain: 1–4 (tune to lighting)
Capture window: 100 ms
```

6.2 Wider View, Mid-FPS

```
ROI: 128 rows, 820 cols (example)
FPS: 500–660
Exposure: 900–1100 µs (ensure Frame_Length ≥ exposure+4)
Capture window: 120 ms
```

6.3 Presentation-Optimized (trimmed)

```
ROI: 64 rows
FPS: 1000
Record length: 60–80 ms
Interpolation(optional): ×2
```

---

7) Installation & Bring-Up (Step-by-Step)

1. Tie grounds at SENSE4 Pin 3.
2. Wire A3 -> divider -> **GPIO17**; verify ~3.2 V at Pi when A3 HIGH.
3. Mount & rotate IMX219 by 90°. Confirm lens focus & ROI coverage.
4. Boot Pi; install packages; deploy modified `raspiraw` and Python scripts.
5. Run `test_signal.py` to confirm clean rising-edge detection on GPIO17.
6. Dry-run capture using `edge_detect_signal.py` (fake/forced trigger) to validate file outputs.
7. Perform a live droplet/UV run; confirm `/dev/shm` outputs and HDMI playback.

---

8) Validation & Acceptance Tests

Electrical - Divider output 3.0–3.3 V on A3 HIGH; <0.8 V on LOW. - Pulse width ≥ 5 ms; no chatter.

Timing - Achieves ≥ 900 fps at ROI=64 with ≤2 dropped frames per 100 ms run. - Visible droplet dwell ≈ 38–40 ms within the UV region.

Function - On trigger, capture starts within ≤ 10 ms. - Files appear in `/dev/shm`.

Stress - 30 back-to-back runs without crash; temperature within safe range; no fps drift.

---

9) Operations

- **Reset policy:** if capture fails (no frames), auto-retry once; then prompt for service.
- **Log files:** `/var/log/exhibit-cam/*.log` for trigger/capture/playback events.
- **Safe shutdown:** power UV off before any software restart.

---

10) Appendix: Reference Snippets

10.1 `raspiraw` example with timestamps

```
./raspiraw -md 7 -t 100 -ts /dev/shm/tstamps.csv -hd0 /dev/shm/hd0.32k \
  -h 64 -w 640 --fps 1000 -sr 1 --regs "0171,01;0170,01" -g 1 -eus 750 \
  -o /dev/shm/out.%06d.raw
```

10.2 Minimal GPIO wait (Python)

```python
import RPi.GPIO as GPIO, subprocess, time
PIN=17
GPIO.setmode(GPIO.BCM); GPIO.setup(PIN, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
while True:
    GPIO.wait_for_edge(PIN, GPIO.RISING)
    subprocess.run([
        "./raspiraw","-md","7","-t","100","--fps","1000",
        "-h","64","-w","640","-g","1","-eus","750",
        "-ts","/dev/shm/tstamps.csv","-o","/dev/shm/out.%06d.raw"
    ])
    subprocess.run(["python3","player_pin.py"])  # or your renderer
```

10.3 Arduino trigger (concept)

```cpp
const int TRIG=A3; void setup(){ pinMode(TRIG, OUTPUT); digitalWrite(TRIG, LOW);}
void loop(){
  if(/* droplet detected */){
    digitalWrite(TRIG, HIGH);   // start capture
    // UV on ...
    delay(50);
    // UV off ...
    digitalWrite(TRIG, LOW);    // done
  }
}
```

---

11) FAQ

- **Q:** Can we move the trigger to a different Pi pin?
  A: Yes, but update the daemon/script and retain *rising-edge* semantics.

- **Q:** Can we widen the view for a bigger scene?
  A: Increase ROI rows (e.g., 128/192). Expect fps to drop roughly inversely with ROI height.

- **Q:** How do we make playback smoother?
  A: Use the interpolation player at ×4; it inserts in-between frames for display.

End of API