Event Driven Industrial Sensor Gateway ASIC (EDISGA)

Cycle-Accurate RTL-Level Architectural Simulation Report

Sudip Roy

February 13, 2026

Abstract

This report presents the design and cycle accurate, RTL style simulation of a synthesizable event driven industrial sensor gateway ASIC. The system operates on a fixed sampling period $T_s = 100\,\text{ms}$ and executes a deterministic per tick schedule comprising multi channel sensor acquisition with oversampling, fixed point quantization in Q8.8, moving average filtering, plant control via a clocked finite state machine (FSM), alarm detection using hysteresis and N tick debounce, bounded buffering in a finite depth historian FIFO, and packetization over a backpressure limited streaming interface using a valid and ready handshake.

The datapath is expressed using finite state and bounded memory primitives consistent with synthesizable RTL: explicit registers, fixed width arithmetic with saturations, circular buffers, and finite queues. Quantitative evaluation is reported through figures and tabular inserts embedded directly in this document: raw versus filtered sensor trajectories, FSM timelines, actuator traces, alarm masks and rising edges, FIFO occupancy and drop behavior, protocol multiplexing behavior, throughput under backpressure, and end to end latency histograms for periodic and event records. Error detection is provided by CRC16 CCITT computed over frame headers and payloads; configuration parameters and validation vectors are included for reproducibility.

Contents

1 Introduction

Industrial edge gateways increasingly sit between noisy physical processes and deterministic digital backends such as historians, supervisory systems, and asset monitoring stacks. A practical ASIC gateway must be deterministic, explainable, and verifiable. It must also remain synthesizable: state is held in explicit registers, arithmetic is bounded and fixed width, memory structures are finite, and handshake interfaces represent real throughput limits.

EDISGA is a cycle accurate, RTL style simulation that models such a gateway on an explicit time base: a 100 ms tick. Each tick executes a structured schedule: sample sensors (with optional oversampling), update fixed point filtering, step a plant control FSM, evaluate alarm comparators with debounce and hysteresis, push fixed width records into a bounded FIFO, and packetize FIFO head records into a streaming interface subject to valid and ready backpressure. The simulation produces waveforms, figures, and configuration metadata that are incorporated in this report as embedded plots and verbatim tables.

The intent of the report is to connect module level RTL semantics with quantitative results such as latency histograms, FIFO occupancy excursions, and CRC validation vectors, without requiring access to the codebase.

2 System Architecture

2.1 Top Level Module Decomposition

A hardware realization is naturally described as a set of RTL modules connected by synchronous interfaces. The simulation aligns to the following module graph:

- **SensorSampler:** Inputs are process dependent sensor values; outputs are raw Q8.8 samples.
- **MovingAverageFilter:** Per channel moving average with window W.
- **DryerFSM:** Takes filtered sensor values and internal dwell counters; outputs actuator control signals.
- **AlarmEventManager:** Compares filtered sensor values to thresholds; implements debounce; outputs alarm mask and edges.
- **HistorianLogger:** Constructs fixed width records; pushes into a bounded FIFO.
- **Packetizer:** Assembles frames with SYNC, header, payload, and CRC16.
- **ProtocolMux:** Selects between two header formats based on a runtime schedule.

2.2 Architectural Block Diagram

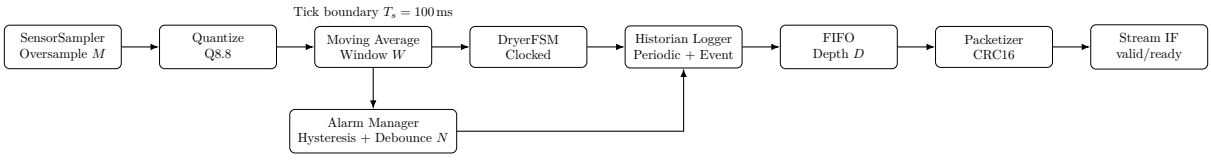

Figure 1: Top level architecture with tick driven datapath and event coupling.

2.3 Record and Frame Interfaces

The internal payload is a fixed width record. Externally the packetizer streams bytes. In RTL terms:

- FIFO write port: `wr_en`, `wr_data[255:0]`, explicit drop on full.
- FIFO read port: `rd_valid` when occupancy non zero, `rd_data`, `rd_pop` asserted when frame transmission completes.
- Stream port: `valid`, `ready`, `data[7:0]`.

2.4 Simulation Generated Record Layout

Figure 2: Historian record layout (fixed width payload).

```
Record layout (fixed width payload)
===================================

ts_ms: u32 | timestamp in milliseconds
fsm_state: u8 | FSM state id
proto_id: u8 | protocol id
alarms_mask: u16 | alarm latch bitfield
temp_q88: i16 | temperature in Q8.8
moist_q88: i16 | moisture in Q8.8
press_q88: i16 | pressure in Q8.8
vib_q88: i16 | vibration in Q8.8
heater_en: u8 | heater enable
fan_pwm: u8 | fan PWM (0..255)
valve: u8 | valve state
event_type: u8 | 0=periodic, 1=alarm_edge
event_code: u16 | event code
fifo_occ: u16 | fifo occupancy snapshot
rsv0: u32 | reserved
rsv1: u32 | reserved
```

3 Methodology and RTL Style Modeling Approach

3.1 Deterministic Scheduling and Tick Semantics

Primary tick period: $T_s = 100\,\text{ms}$. This is treated as a synchronous cycle boundary for all sequential state. Within one tick, the model executes a fixed ordering of operations:

1. Sensor sampling with optional oversampling (subsamples) and averaging.
2. Fixed point conversion to Q8.8 integers.
3. Per channel moving average filtering implemented as an integer accumulator and circular buffer.
4. Control FSM step producing actuator outputs.
5. Alarm manager step evaluating latched alarms using debounce counters and set clear thresholds.
6. Historian record generation: one periodic record per tick and an additional record on any alarm rising edge.
7. FIFO push of generated records; explicit drop on overflow.
8. Packetizer step: if idle and FIFO non empty and ready asserted, construct a frame for FIFO head; if active and ready asserted, transmit up to a per tick byte budget; when complete, pop FIFO and account for latency.

This schedule resembles a synthesizable multi stage pipeline with explicit combinational and sequential boundaries.

3.2 Fixed Width Arithmetic and Saturations

Sensor values are represented in Q8.8 fixed point: a signed integer encodes a real value x as

$$x_q = \mathrm{round}(x \cdot 2^8), \qquad \hat{x} = x_q \cdot 2^{-8}.$$

Quantization error is bounded by $|e_q| \leq 2^{-9}$ in real units. For i16 record storage, values are saturated to $[-32768, 32767]$ prior to packing.

3.3 Synthesizable Memory Structures

The historian FIFO is a finite depth circular buffer with explicit read and write pointers and an occupancy counter. Push drops on full and increments a drop counter. Peek is used to start packetization without committing a pop until the full frame is transmitted. This models a design in which payload memory and the streaming datapath are decoupled but record lifetime is controlled by FIFO semantics.

4 Sampling Theory and Timestamp Resolution

4.1 Sampling Period and Effective Rate

Primary tick: $T_s = 100\,\mathrm{ms}$, therefore $f_s = 10\,\mathrm{Hz}$. Oversampling uses M subsamples per tick:

$$\bar{x}[k] = \frac{1}{M} \sum_{m=0}^{M-1} x\left(kT_s + m\frac{T_s}{M}\right),$$

followed by quantization to Q8.8.

4.2 Timestamp Model and Bounded Jitter

Each record stores `ts_ms` in milliseconds. The model sets:

$$\texttt{ts\_ms}[k] = k \cdot T_s.$$

A bounded jitter model for a physical gateway can be written as

$$t_k = kT_s + \epsilon_k, \quad |\epsilon_k| \leq \epsilon_{\max},$$

with $\epsilon_k = 0$ in the simulation by construction.

5 Fixed Point Filtering and Quantization Error

5.1 Moving Average Filter

Moving average window length W. Synthesizable implementation uses circular buffer, accumulator $S[k]$, and the recurrence:

$$S[k] = S[k-1] + x[k] - x[k-W], \qquad y[k] = \left\lfloor \frac{S[k]}{W} \right\rfloor.$$

5.2 Impulse Response and Group Delay

The moving average has an effective group delay of approximately $(W-1)/2$ ticks:

$$\tau_g \approx \frac{W-1}{2}\, T_s.$$

For example, $W = 8$ yields $\tau_g \approx 3.5 \cdot 0.1\,\mathrm{s} = 0.35\,\mathrm{s}$.

5.3 Quantization Noise Bounds

For Q8.8, the quantization error satisfies $|e_q| \leq 2^{-9}$. The moving average output additionally incurs truncation from integer division.

5.4 Synthesizable Datapath View of the Moving Average

The moving average filter is implemented using a subtract add accumulator and a circular buffer indexed by `idx`. At each tick, the newest quantized sample $x[k]$ is added to the accumulator, while the oldest sample $x[k-W]$ stored in the buffer is subtracted. This structure avoids recomputing the full window sum and yields constant time per update.

The division by W is modeled as an integer division with floor semantics. When W is a power of two, this operation may be realized as a right shift in hardware; otherwise, it corresponds to a small divider or reciprocal multiply depending on design constraints. In this simulation, W is a configurable integer and division is exact integer division.

All state elements are explicit and bounded: the accumulator register $S[k]$, the circular buffer of length W, and the buffer index `idx`. The datapath is fully synthesizable and maps directly to a single clocked RTL process.

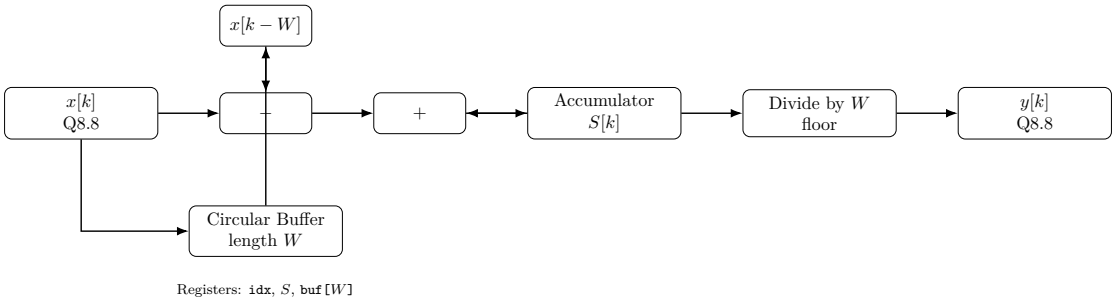

Figure 3: Synthesizable moving average datapath using subtract add accumulation and a circular buffer.

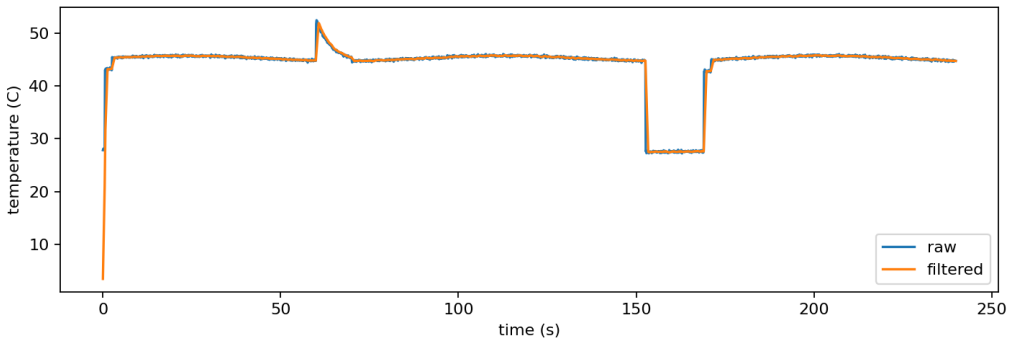

Figure 4: Temperature raw vs filtered output (Q8.8 interpreted as Celsius).

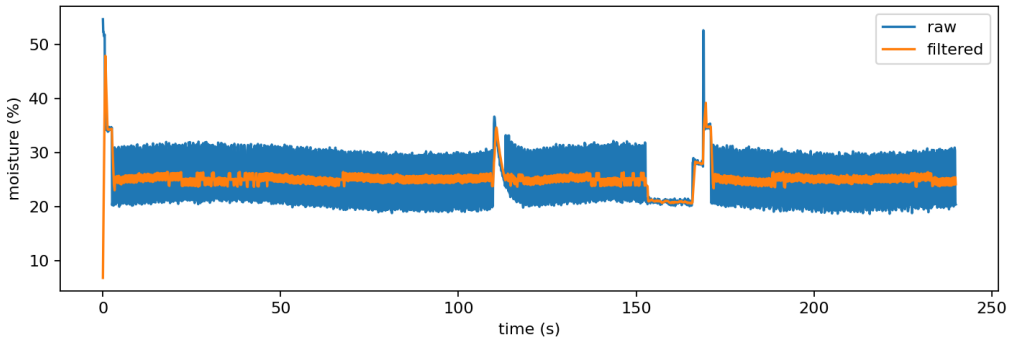

Figure 5: Moisture raw vs filtered output (Q8.8 interpreted as percent).

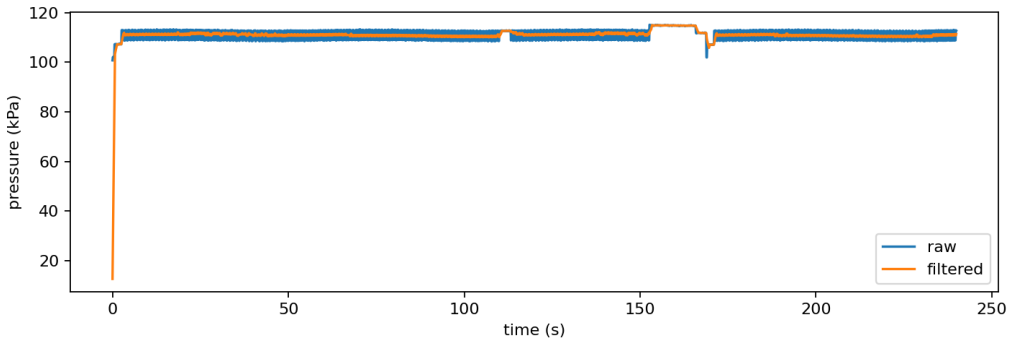

Figure 6: Pressure raw vs filtered output (Q8.8 interpreted as kPa).

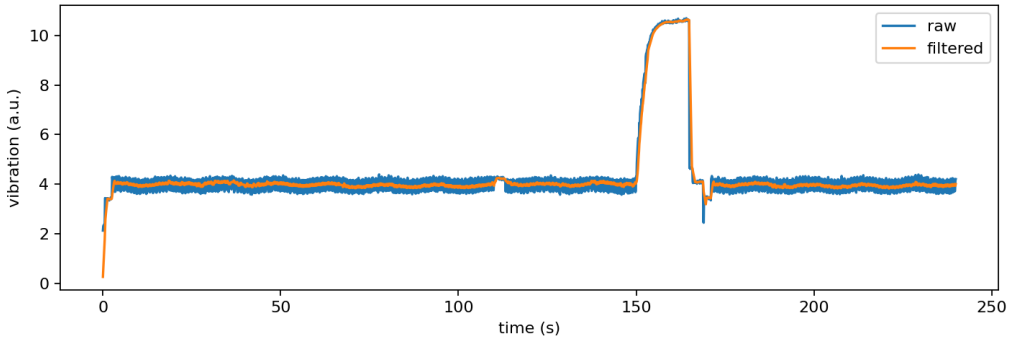

Figure 7: Vibration raw vs filtered output (Q8.8 interpreted as arbitrary units).

6 FSM Design and Timing Behavior

6.1 State Encoding and Transitions

Control logic is a clocked FSM with five states: IDLE, RAMP, STEADY, FAULT, COOLDOWN. Transitions are driven by filtered sensor values and dwell counters.

6.2 Actuator Outputs and Safety

Actuator outputs are Mealy style, depending on state and internal counters. Fault transitions have highest precedence; if a fault condition is detected, the state transitions to FAULT immediately.

6.3 State Diagram (Control FSM)

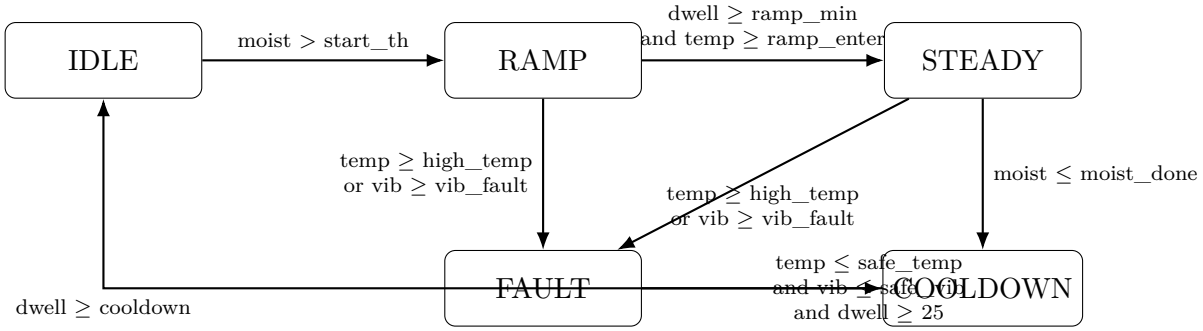

Figure 8: Dryer control FSM state diagram (conditions match the simulation logic).

6.4 FSM as a Synchronous Next-State Function

A synthesizable FSM is typically expressed as:

$$s[k+1] = \delta\left(s[k],\, x[k]\right), \qquad u[k] = \lambda\left(s[k],\, x[k]\right),$$

where s is the discrete state register, x are filtered sensor inputs, and u are actuator outputs. The dwell counter is an additional state register updated per tick:

$$d[k+1] = \begin{cases} 0, & \text{if state transition occurs} \\ d[k]+1, & \text{otherwise} \end{cases}$$

6.5 Actuator Output Logic (Interpretation)

The simulation logic yields actuator behaviors that can be described as:

- IDLE: low fan, heater off, valve closed.
- RAMP: heater on, fan medium, valve closed.
- STEADY: fan high; heater is duty-modulated in-band; valve depends on moisture threshold.
- FAULT: heater off; fan max; valve open.
- COOLDOWN: heater off; fan high; valve open; return to IDLE after cooldown dwell.

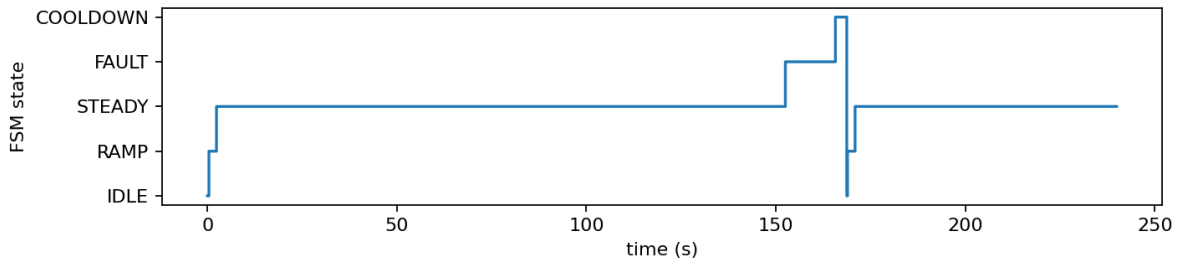

Figure 9: FSM state timeline at tick resolution.

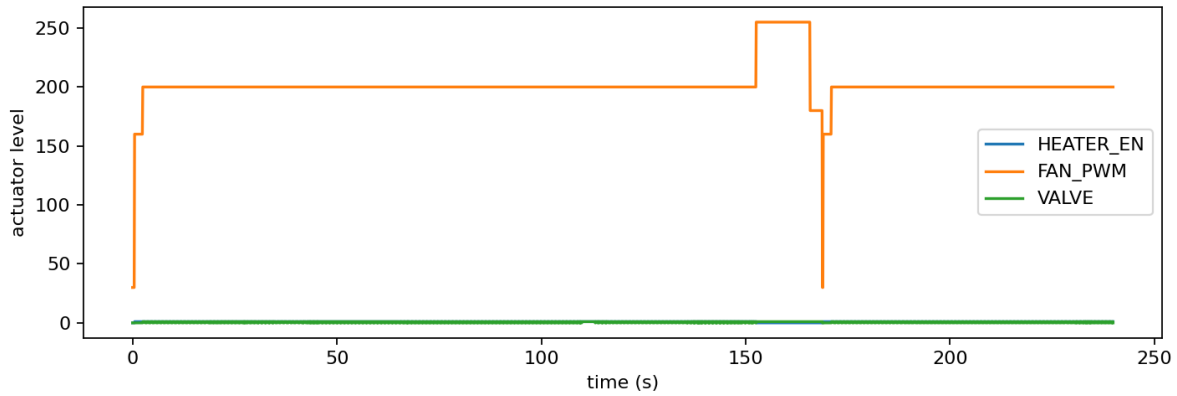

Figure 10: Actuator outputs driven by the control FSM.

7 Alarm Debounce Derivation

7.1 Hysteresis and Debounce

Each alarm is a latched comparator with separate set and clear thresholds. If unlatched and the set condition holds for N consecutive ticks, the latch becomes 1. If latched and the clear condition holds for N consecutive ticks, the latch becomes 0. With $T_s = 100\,\text{ms}$ and $N = 3$, the minimum persistent excursion required is $NT_s = 300\,\text{ms}$.

7.2 Debounce as a Small Two-State FSM

The debouncer can be viewed as a two-state FSM per alarm bit with an N-tick counter. A compact model is:

$$\texttt{cnt}[k+1] = \begin{cases} \texttt{cnt}[k] + 1, & \text{if condition holds} \\ 0, & \text{otherwise} \end{cases}$$

and the latch toggles when $\texttt{cnt} \geq N$ under the appropriate set or clear condition.

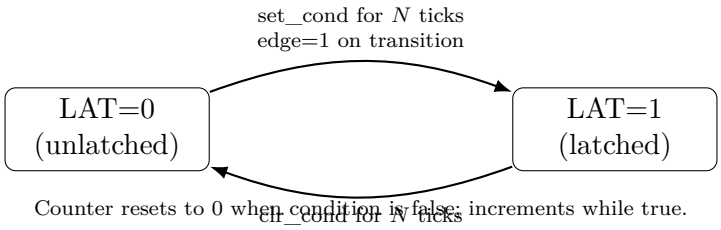

Figure 11: Debounced hysteresis latch interpreted as a two-state FSM with an N-tick counter.

7.3 Event Detection

An alarm event is defined as a rising edge (latch transition 0 to 1). A priority encoder assigns a single event code per tick in the presence of multiple edges.

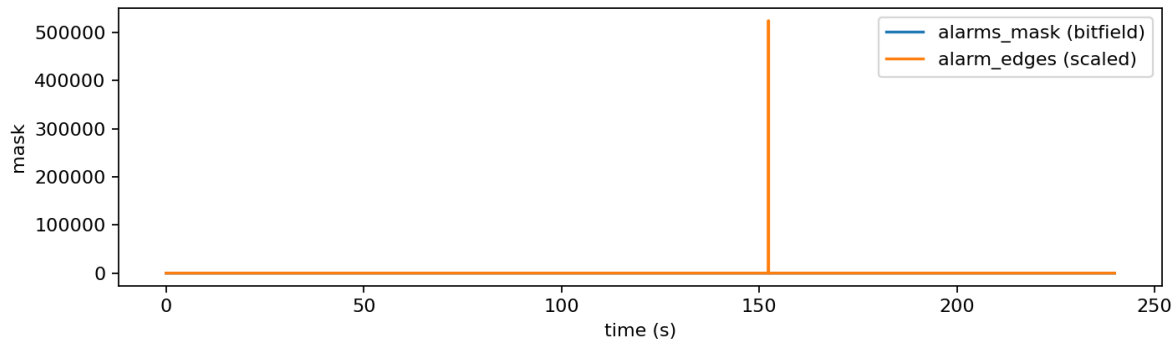

Figure 12: Alarm latch outputs and rising edge detections.

8 Historian Logger and FIFO Sizing

8.1 Record Generation Rate

Attempted record rate is $R_{\text{in}} = R_p + R_e$, where $R_p = 1/T_s = 10\,\text{Hz}$ is the periodic rate and R_e is the event driven increment. Let $A[k]$ be the number of records attempted at tick k (1 or 2), and let $D[k]$ be the number of records dequeued for transmission completion at tick k (0 or 1). The occupancy recurrence is:

$$O[k+1] = \min(D_{\text{FIFO}},\, O[k] + A[k] - D[k]),$$

with explicit drops on overflow when $O[k] = D_{\text{FIFO}}$ and $A[k] > D[k]$.

8.2 FIFO Circuit Sketch (Synthesizable View)

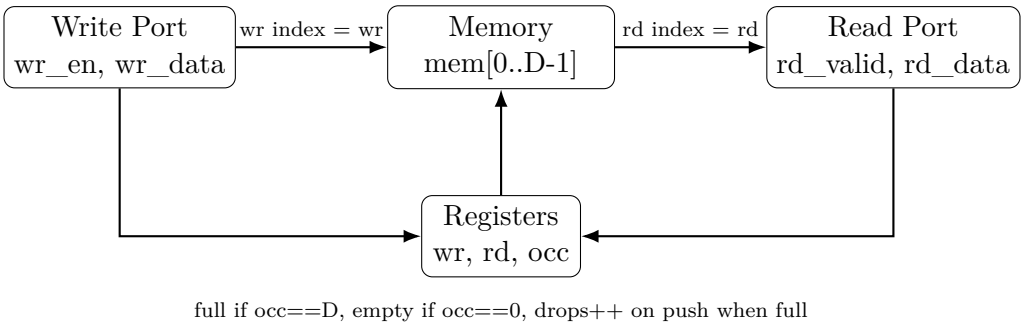

Figure 13: Historian FIFO modeled as circular memory with pointers and occupancy register.

9 Packetizer and CRC Polynomial

9.1 Frame Format and CRC16

Frame format:

`[SYNC] [HEADER] [PAYLOAD] [CRC16].`

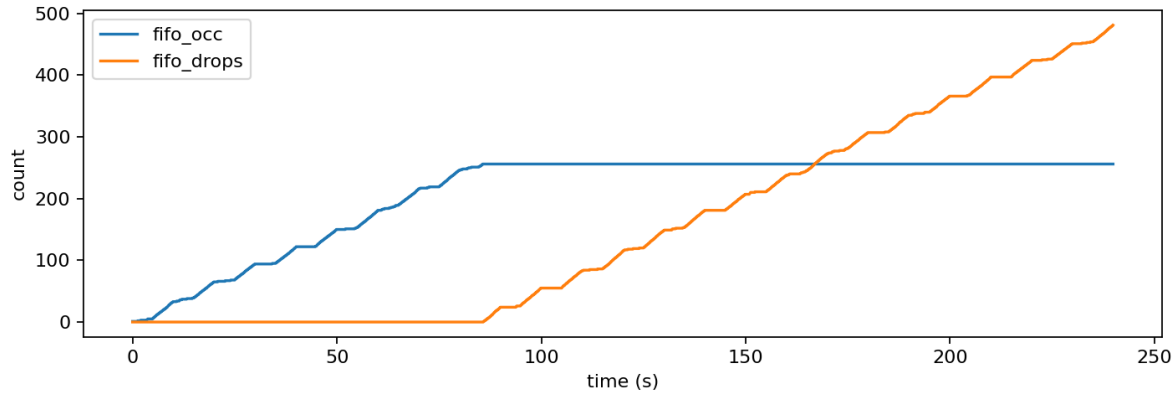

Figure 14: FIFO occupancy and record drops over time.

SYNC bytes are `0xA5 0x5A`. CRC16 CCITT polynomial over GF(2):

$$G(x) = x^{16} + x^{12} + x^5 + 1.$$

CRC is computed over `HEADER || PAYLOAD` (SYNC excluded), using the parameters embedded in the configuration.

9.2 CRC Bitwise Update as an LFSR (Circuit Perspective)

A common RTL implementation uses a 16-bit shift register with conditional XOR taps according to $G(x)$. In the non-reflected CCITT form used here, each input bit is processed MSB-first with a feedback bit. The simulation uses the exact same bitwise behavior.

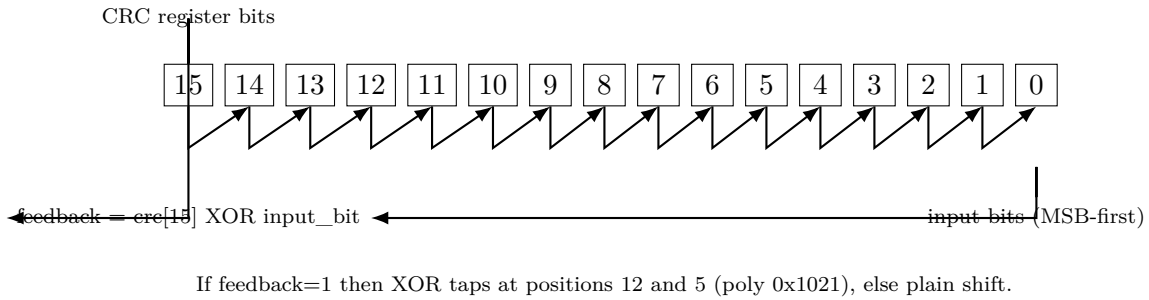

Figure 15: CRC16-CCITT modeled as an LFSR-like shift register with conditional XOR taps (conceptual circuit sketch).

9.3 Embedded CRC Configuration and Validation Vectors

Figure 16: CRC configuration (embedded JSON).


```
{
  "init_hex": "0xFFFF",
  "poly_hex": "0x1021",
  "refin": false,
  "refout": false,
  "scope": "CRC computed over header + payload (SYNC excluded)",
  "width": 16,
  "xorout_hex": "0x0000"
}
```


Figure 17: CRC validation vectors (embedded CSV excerpt).


```
proto_id,header_hex,payload_prefix_hex,crc16_hex
0,11032000,00000000000000007a03d506...,0xE50D
0,11032000,bc02000001000000b71fe62f...,0x3938
0,11032000,7805000001000000272b4522...,0xD28F
0,11032000,3408000001000000302b4e22...,0x0230
0,11032000,f00a000002000000f62b391d...,0x6207
0,11032000,ac0d000002000000522d7c19...,0x4304
```


10 Protocol Multiplexer

Two protocol header formats differ structurally but share payload and CRC mechanism:

- **Modbus-like:** `[addr][func][len_lo][len_hi]` (4 bytes)
- **CIP-like:** `[service][class][instance][len_lo][len_hi]` (5 bytes)

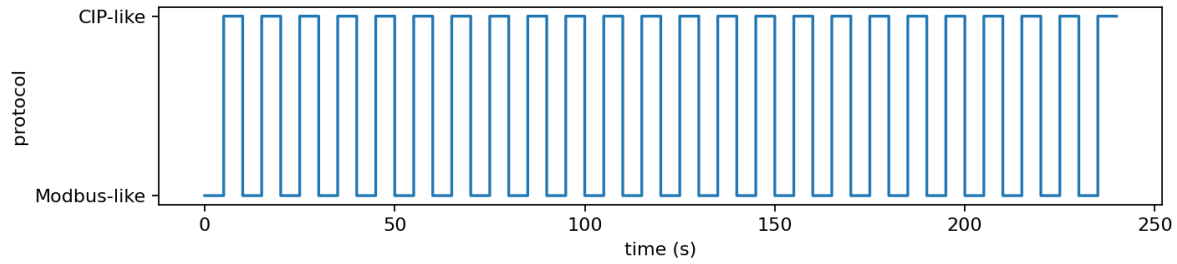

Figure 18: Protocol multiplexer selection over time.

11 Streaming Handshake, Throughput, and Latency Bounds

11.1 Handshake Model

The valid and ready handshake models streaming output under backpressure. A frame starts only when the packetizer is idle, FIFO is non empty, and ready is high. When active, the packetizer transmits up to a per tick byte budget while ready remains asserted.

11.2 Packetizer Micro-FSM (Idle vs Sending)

The packetizer is naturally modeled as a two-state FSM:

- IDLE: if FIFO non-empty and ready=1, capture header and payload, compute CRC, start sending, assert start_fire.
- SENDING: each tick, if ready=1, transmit up to B bytes; when frame bytes complete, pop FIFO and return to IDLE.

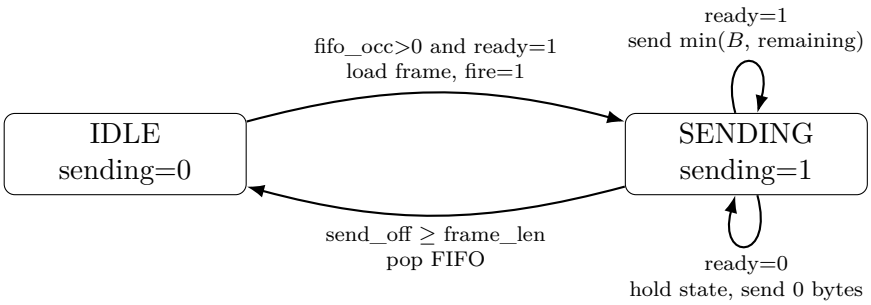

Figure 19: Packetizer micro-FSM controlling start, byte transmission, and FIFO pop.

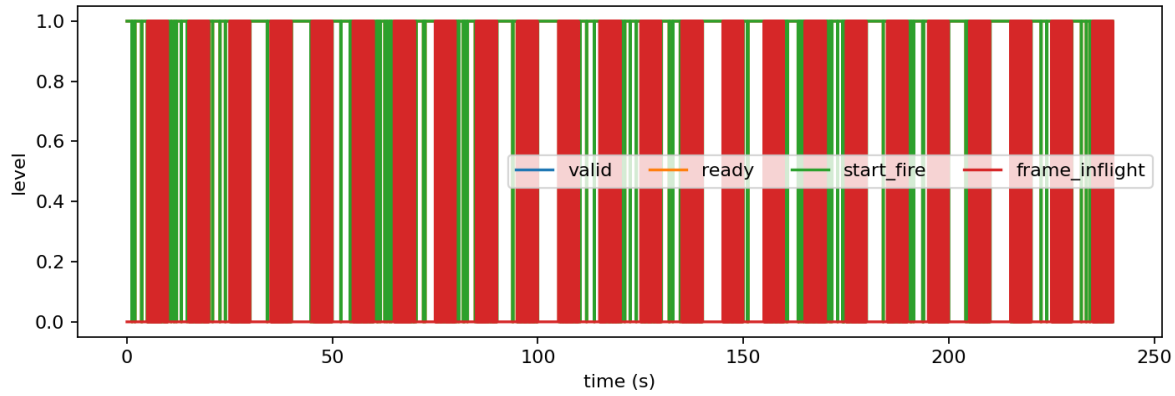

Figure 20: Valid and ready handshake behavior.

11.3 Backpressure Adjusted Service Bound

Let ρ denote the fraction of ticks where ready is high. Let B be the byte budget per tick and let L_f be the frame length in bytes. A lower bound on the number of ticks required per frame is $\lceil L_f/B \rceil$, so a crude service rate bound is

$$R_{\text{svc}} \lesssim \frac{\rho}{\lceil L_f/B \rceil} \cdot \frac{1}{T_s}.$$

When $R_{\text{in}} > R_{\text{svc}}$ in expectation, FIFO occupancy grows toward capacity, leading to drops.

11.4 Latency Definition and Measurement

Latency is measured in ticks from record enqueue to record dequeue (pop) when the frame completes transmission. If a record is enqueued at tick k_e and popped at tick k_p, then:

$$L = k_p - k_e.$$

Separate latency series are tracked for periodic records and event records.

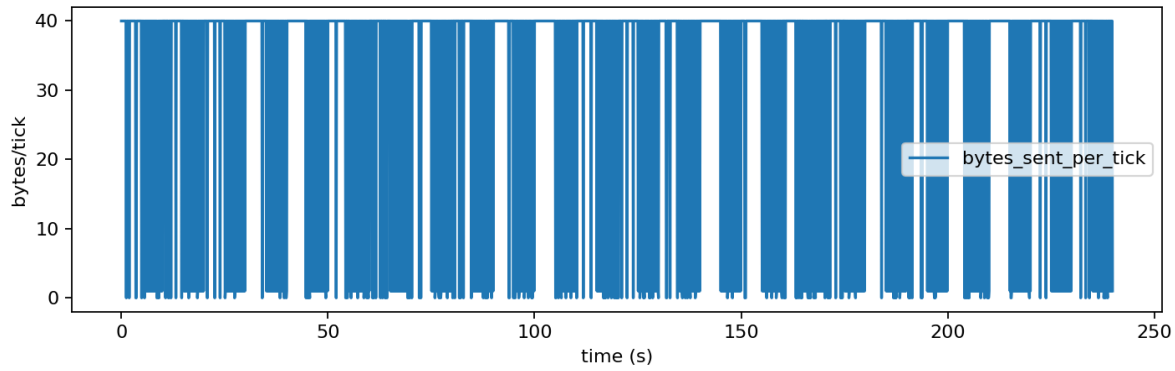

Figure 21: Bytes transmitted per tick under backpressure.

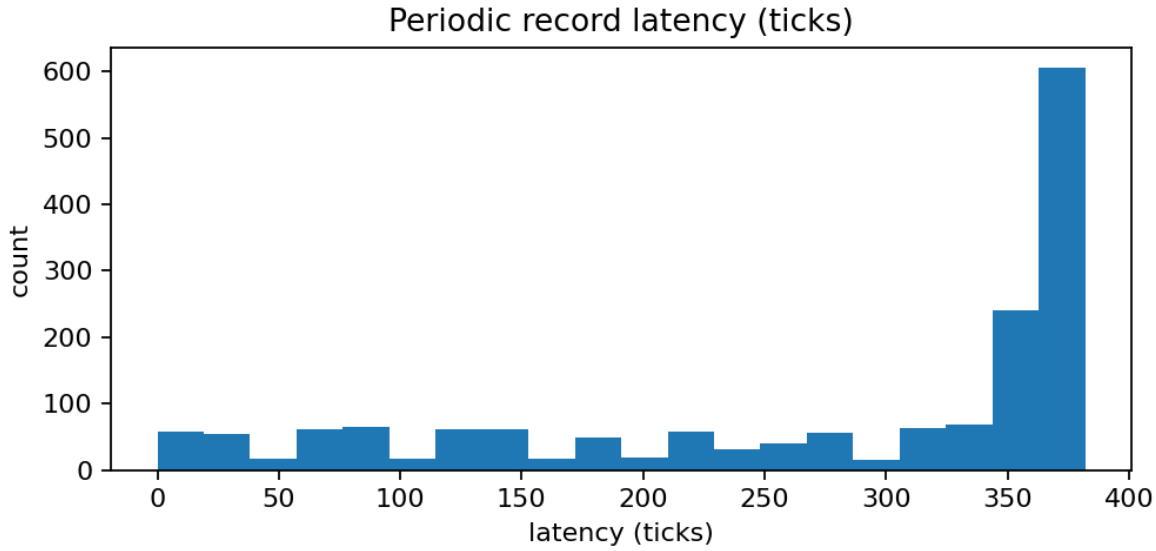

Figure 22: Latency histogram for periodic records (ticks).

12 Verification Strategy and Determinism

The simulation uses a fixed random seed to produce deterministic noise processes. With identical seed and configuration, outputs are reproducible.

12.1 Pseudo RTL Listings

Listing 1: Synthesizable Moving Average Update (Q8.8)

```
/* State: int32 acc; int16 buf[W]; int idx; */
old = buf[idx];
acc = acc - old + x_q88;
buf[idx] = x_q88;
idx = (idx + 1) % W;
y_q88 = (int16)(acc / W); /* integer division */
```

Listing 2: Debounced Latch with Hysteresis

```
/* State: bit latched; uint cnt; */
edge = 0;
if (!latched) {
  if (set_cond(x_q88)) {
    cnt++;
    if (cnt >= N) { latched = 1; cnt = 0; edge = 1; }
  } else cnt = 0;
} else {
  if (clr_cond(x_q88)) {
    cnt++;
    if (cnt >= N) { latched = 0; cnt = 0; }
  } else cnt = 0;
}
```

Listing 3: CRC16 CCITT Bitwise Update

```
crc = 0xFFFF;
for each byte b in (header || payload) {
  crc ^= ((uint16)b << 8);
  repeat 8 times {
    if (crc & 0x8000) crc = (crc << 1) ^ 0x1021;
    else              crc = (crc << 1);
    crc &= 0xFFFF;
  }
}
```

Listing 4: FIFO Push Pop Semantics

```
/* State: mem[D], rd, wr, occ, drops */
push(item):
  if (occ == D) { drops++; return 0; }
  mem[wr] = item; wr = (wr + 1) % D; occ++; return 1;

peek():
  if (occ == 0) return (0, null);
  return (1, mem[rd]);

pop():
  if (occ == 0) return (0, null);
  item = mem[rd]; rd = (rd + 1) % D; occ--; return (1, item);
```

13 Results and Discussion

13.1 Sensor Dynamics

The sensor generator includes deterministic disturbance windows that create excursions in temperature, vibration, and moisture, permitting observation of filtering, control response, and alarm behavior.

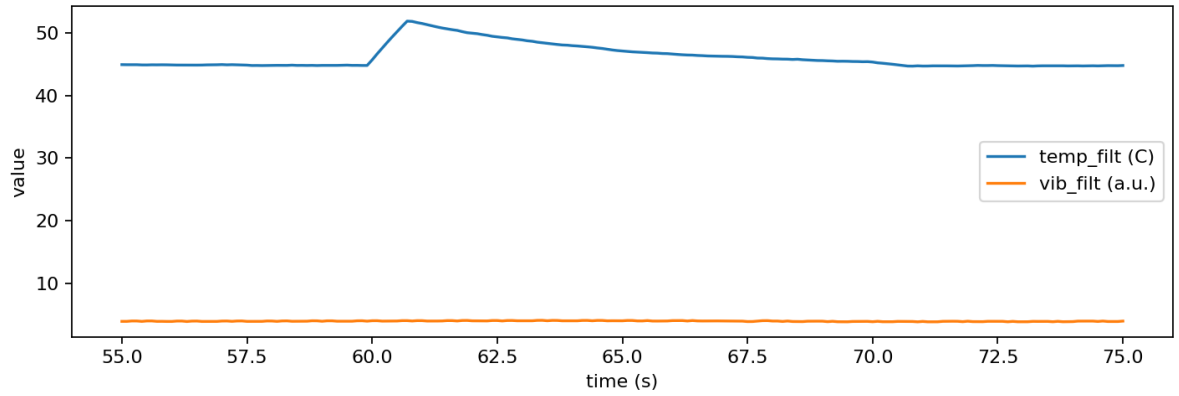

Figure 23: Zoom: Temperature drift and vibration disturbance.

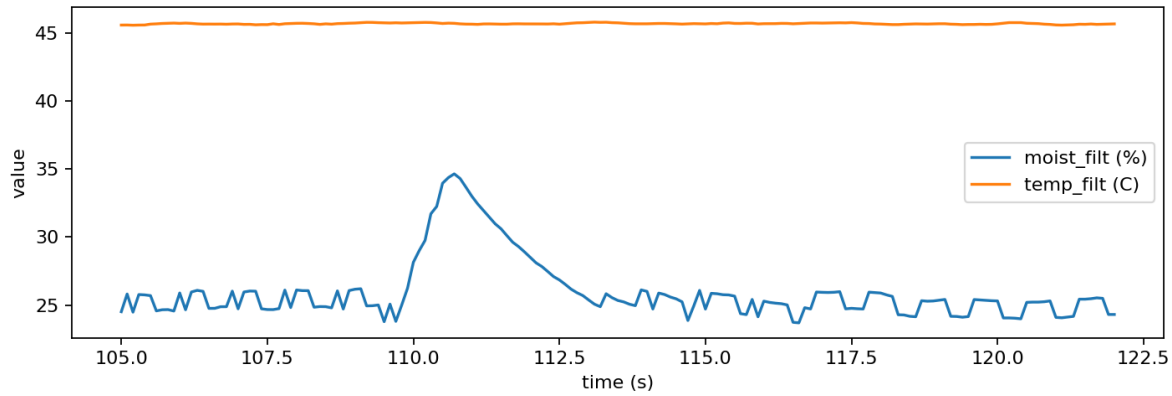

Figure 24: Zoom: Moisture transient and control response.

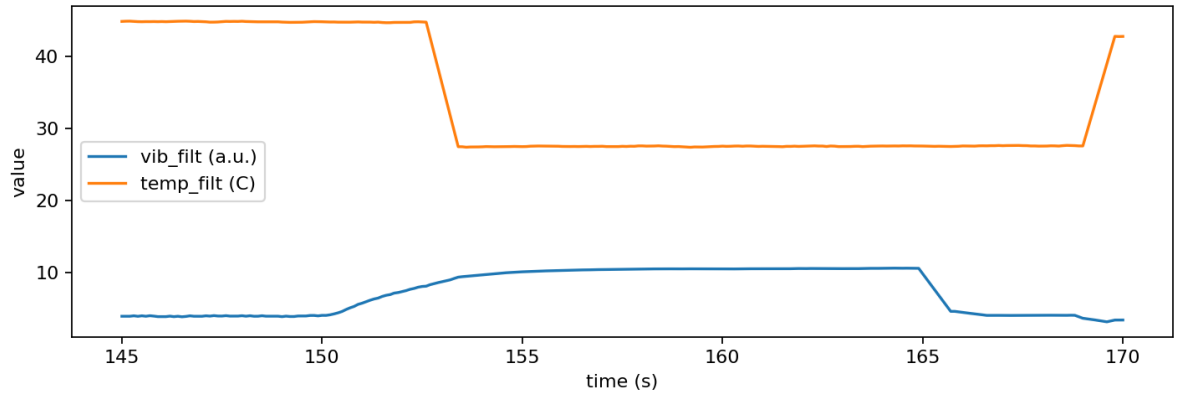

Figure 25: Zoom: Vibration disturbance segment and system response.

13.2 Run Summary (Embedded)

Figure 26: Run summary (embedded JSON).


```
{
  "average_throughput_Bps": 279.825,
  "event_records_attempted": 1,
  "fifo_drops_records": 481,
  "frames_emitted": 1665,
  "latency_ticks": {
    "event": {
      "max": 0.0,
      "mean": 0.0,
      "min": 0.0,
      "n": 0,
      "p95": 0.0
    },
    "periodic": {
      "max": 382.0,
      "mean": 272.13933933933936,
      "min": 0.0,
      "n": 1665,
      "p95": 380.0
    }
  },
  "records_attempted": 2401,
  "ticks": 2400,
  "ticks_with_backpressure_ready0": 177,
  "total_bytes_transmitted": 67158
}
```

13.3 Configuration Metadata (Embedded)

Figure 27: Configuration metadata (embedded CSV).

```
parameter,value
Project,Event Driven Industrial Sensor Gateway ASIC (EDISGA)
Seed,7
Tick period,100 ms
Duration,240 s
Total ticks,2400
Oversampling M,8
Moving average window W,8
FIFO depth (records),256
Byte budget per tick,40
Protocol switch period (ticks),50
CRC16 polynomial,0x1021 (CCITT)
CRC16 init,0xFFFF
Record bytes,32
Frame SYNC,a55a
```

13.4 Thresholds (Embedded)

Figure 28: Alarm thresholds (embedded CSV).

```
signal,threshold
HIGH_TEMP set,95.0 C
HIGH_TEMP clear,90.0 C
LOW_MOISTURE set,12.0 %
LOW_MOISTURE clear,16.0 %
HIGH_PRESSURE set,140.0 kPa
HIGH_PRESSURE clear,132.0 kPa
HIGH_VIBRATION set,7.50 a.u.
HIGH_VIBRATION clear,6.00 a.u.
```

14 Conclusion

EDISGA demonstrates a complete event driven sensor gateway datapath consistent with synthesizable logic: tick based sampling, fixed point filtering, a clocked control FSM, debounced alarm latching with edge capture, deterministic record logging into a bounded FIFO with explicit overflow, and packetization with CRC into a valid and ready streaming interface under throughput constraints. The embedded figures and tables provide trace level evidence for control behavior, alarm dynamics, buffering effects under backpressure, and integrity checks via CRC configuration and validation vectors.