Energy Harvesting Simulator - Technical Documentation

October 26, 2025

Contents

---

1 System Overview

1.1 System Architecture

The simulator models a solar-powered IoT device with the following components:

```
Solar Panel → Harvested Energy (E_harv)

      ↓

   PMIC     → Efficiency losses

      ↓

Supercap    → Energy Storage (E_cap)

      ↓

MCU + Model → Load Energy (E_load)
```

1.2 Operating Modes

Mode	Description	Example conditions
HIGH	Full-quality inference	$V \geq 3.1\,\mathrm{V}$, buffer > 0
DEGRADED	Reduced-quality inference	$V \geq 2.9\,\mathrm{V}$, buffer > 0
SLEEP	Idle with minimal power	$V \geq 2.9\,\mathrm{V}$, buffer $= 0$
SLEEP_CRITICAL	Deep sleep	$V < 2.9\,\mathrm{V}$
SHUTDOWN	System shutdown	$V < 2.6\,\mathrm{V}$

---

2 Mathematical Models

2.1 Supercapacitor Energy and Voltage

Energy stored in capacitor:

$$E_{\text{cap}} = \tfrac{1}{2}CV^2$$

Where:

- E_{cap} = Energy stored (Joules)

- C = Capacitance (Farads)
- V = Voltage across capacitor (Volts)

Voltage from energy:

$$V = \sqrt{\frac{2E_{\text{cap}}}{C}}$$

Energy update per timestep:

$$\boxed{E_{\text{cap}}(t + \Delta t) = E_{\text{cap}}(t) + E_{\text{harv}}(\Delta t) - E_{\text{load}}(\Delta t)}$$

2.2 Solar Energy Harvesting

Instantaneous power from photovoltaic panel:

$$P_{\text{pv}}(t) = \eta_{\text{pv}}\, A_{\text{pv}}\, G(t)$$

Where:

- P_{pv} = Power output (Watts)
- η_{pv} = Panel efficiency (dimensionless, 0-1)
- A_{pv} = Panel area (m^2)
- $G(t)$ = Solar irradiance at time t (W/m^2)

Energy harvested over timestep:

$$\boxed{E_{\text{harv}}(\Delta t) = P_{\text{pv}}(t) \times \Delta t \times \eta_{\text{PMIC}}}$$

Where:

- η_{PMIC} = PMIC efficiency (typically 0.80–0.90)
- Δt = Timestep duration (seconds)

2.3 Load Energy Consumption

Energy per inference/action:

$$\boxed{E_{\text{load}} = V_{\text{supply}} \times I_{\text{mode}} \times T_{\text{mode}}}$$

Where:

- V_{supply} = Supply voltage (V)
- I_{mode} = Current draw (A)
- T_{mode} = Execution time (s)

2.4 Buffer Dynamics

Buffer occupancy update:

$$\boxed{B(t + \Delta t) = B(t) + r_{\text{in}} \times T_{\text{action}} - n_{\text{processed}}}$$

Where:

- $B(t)$ = Buffer occupancy at time t (frames)
- r_{in} = Input frame rate (frames/second)
- T_{action} = Time elapsed during action (s)
- $n_{\text{processed}}$ = Number of frames processed (0 or 1)

Constraints:

$$0 \leq B(t) \leq B_{\text{max}}$$

Where B_{max} = buffer capacity (typically 10 frames)

Buffer overflow condition:

$$\text{If } B(t) \geq B_{\text{max}} \rightarrow \text{oldest frames are dropped}$$

2.5 Decision Logic (Rule-Based Controller)

Action is now chosen based on rule-based conditions, next week the Rl controller would be applied here.

2.6 Energy Balance Equation

Complete system energy balance per timestep:

$$E_{\text{cap}}(t + \Delta t) = E_{\text{cap}}(t) + \left[\eta_{\text{pv}} A_{\text{pv}} G(t) \Delta t \eta_{\text{PMIC}}\right] - \left[V_{\text{supply}} I_a(t) T_a(t)\right]$$

Simplified:

$$\Delta E_{\text{cap}} = E_{\text{harv}} - E_{\text{load}}$$

Power balance:

$$P_{\text{net}}(t) = P_{\text{harv}}(t) - P_{\text{load}}(t)$$

Where:

- $P_{\text{net}} > 0 \Rightarrow$ capacitor charging
- $P_{\text{net}} < 0 \Rightarrow$ capacitor discharging
- $P_{\text{net}} = 0 \Rightarrow$ energy neutral

---

3 Simulation Scenarios

3.1 NREL Data Integration

The simulator supports **real-world solar irradiance data** from NREL's National Solar Radiation Database (NSRDB).

NREL Data Format

Expected CSV format (example rows):

```
Year, Month, Day, Hour, Minute, GHI, DNI, DHI, Temperature, ...
2024, 1, 15, 0, 0, 0, 0, 0, 5.2, ...
2024, 1, 15, 0, 10, 0, 0, 0, 5.1, ...
```

Key columns:

- **GHI** (Global Horizontal Irradiance) – W/m^2
- **DNI** (Direct Normal Irradiance) – W/m^2
- **DHI** (Diffuse Horizontal Irradiance) – W/m^2
- **Timestamp** Year/Month/Day/Hour/Minute

Data characteristics:

- Temporal resolution: 10-minute intervals (NSRDB standard)
- Negative values indicate missing/invalid data (quality flags)
- Geographic coverage: global (We used data for the following region: 3762824)

3.2 Automated Scenario Extraction

The simulator automatically identifies representative days from NREL data.

Scenario 1: Sunny Day (Optimal Conditions)

Selection criteria:

- Highest mean daily irradiance
- Lowest standard deviation
- No cloud cover / intermittent shading

Typical characteristics:

- Mean GHI: 400–600 W/m^2
- Peak GHI: 800–1000 W/m^2
- Std dev: $<$100 W/m^2
- Smooth bell curve profile

Expected system behavior:

- Voltage rises above 4.0 V at midday
- Sustained HIGH mode (10–14 hours)
- Minimal DEGRADED or SLEEP periods
- Zero shutdowns
- Energy surplus for overnight operations
- Buffer rarely exceeds 50% capacity

Use cases: baseline performance, charging verification.

Scenario 2: Cloudy Day (Challenging Conditions)

Selection criteria:

- Low mean daily irradiance (bottom 30% percentile)
- High standard deviation (intermittent clouds)
- Frequent fluctuations

Typical characteristics:

- Mean GHI: 100–250 W/m^2
- Peak GHI: 300–500 W/m^2
- Std dev: $>$150 W/m^2
- Irregular jagged profile

Expected behavior: voltage 2.8–3.5 V, frequent switching, possible shutdowns.

Scenario 3: Partly Cloudy (Realistic Variable Conditions)

Selection criteria: medium mean irradiance (40–70%), high variance, mix sunny/cloudy.

Typical characteristics: mean GHI 300–500 W/m^2, peak 600–900 W/m^2.

Expected: voltage swings 2.9–4.0 V, dynamic adaptation, occasional brief shutdowns.

Scenario 4: Winter Day (Seasonal Variation)

Selection criteria: best day from Dec–Feb, accounts for short daylight and lower sun angle.

Typical: mean GHI 200–400 $\mathrm{W/m^2}$, peak 500–700 $\mathrm{W/m^2}$, harvesting window 8–10 h.

Expected: extended nighttime use of stored energy; more DEGRADED mode.

Scenario 5: Summer Day (Peak Performance)

Selection criteria: best day from June–Aug, long daylight, high sun angle.

Typical: mean GHI 500–700 $\mathrm{W/m^2}$, peak 900–1200 $\mathrm{W/m^2}$, harvesting window 14–16 h.

Expected: voltage near 5.5 V upper limit, mostly HIGH mode.

4 Function-to-Equation Mapping

4.1 Core Physics Functions

Function: _compute_harvested_power() Implements:

$$P_{\text{pv}} = \eta_{\text{pv}} \times A_{\text{pv}} \times G(t)$$

```
def _compute_harvested_power(self, irradiance: float) -> float:
    return self.config.pv_efficiency * self.config.pv_area *
       irradiance
```

Inputs: `irradiance` ($\mathrm{W/m^2}$)
Output: `power` (W)

Function: _update_capacitor() Implements:

$$E_{\text{cap,new}} = E_{\text{cap,old}} + E_{\text{harv}} - E_{\text{load}}$$

$$V_{\text{new}} = \sqrt{\frac{2E_{\text{cap,new}}}{C}}$$

```
def _update_capacitor(self, e_harv: float, e_load: float):
    # Energy update
    e_new = self.state.energy + e_harv - e_load

    # Clamp to limits
    e_max = 0.5 * self.config.capacitance * self.config.v_max**2
    e_new = max(0.0, min(e_max, e_new))

    # Voltage from energy
    v_new = np.sqrt(2 * e_new / self.config.capacitance)

    return e_new, v_new
```

Inputs: `e_harv` (J), `e_load` (J)
Outputs: `e_new` (J), `v_new` (V)

Function: _compute_load_energy() Implements:

$$E_{\text{load}} = V_{\text{supply}} \times I_{\text{mode}} \times T_{\text{mode}}$$

```
def _compute_load_energy(self, action: Action):
    if action == Action.HIGH:
        energy = cfg.v_supply * cfg.i_high * cfg.t_high
        time = cfg.t_high
        frames = 1
    elif action == Action.DEGRADED:
        energy = cfg.v_supply * cfg.i_degraded * cfg.t_degraded
        time = cfg.t_degraded
        frames = 1
    # ... etc
```

Input: `action` (Enum)
Outputs: `energy` (J), `time` (s), `frames` (int)

4.2 Buffer Management Functions

Function: _update_buffer() Implements:

$$B_{\text{new}} = B_{\text{old}} + r_{\text{in}} \cdot \text{time\_elapsed} - \text{frames\_processed}$$

```
def _update_buffer(self, time_elapsed: float, frames_processed:
   int):
    # Incoming frames
    frames_in = self.config.r_in * time_elapsed

    # Update buffer
    buffer_new = self.state.buffer + frames_in - frames_processed

    # Clamp to capacity
    buffer_new = max(0.0, min(self.config.buffer_capacity,
        buffer_new))

    return buffer_new
```

Inputs: `time_elapsed` (s), `frames_processed` (count)
Output: `buffer_new` (frames)

4.3 Control Logic Functions

Function: _select_action() Implements selection:

$$a^* = \arg\max \text{priority}(a) \quad \text{s.t. } V \geq V_{\min}(a) \text{ and } B > 0$$

```python
def _select_action(self):
    v = self.state.voltage
    buffer_available = self.state.buffer > 0

    # Priority: HIGH > DEGRADED > SLEEP
    if v >= self.config.v_high_safe and buffer_available:
        return Action.HIGH
    elif v >= self.config.v_degraded_safe and buffer_available:
        return Action.DEGRADED
    elif v >= self.config.v_warn:
        return Action.SLEEP
    else:
        return Action.SLEEP_CRITICAL
```

Inputs: `state.voltage`, `state.buffer`
Output: `action` (Enum)

Function: _check_shutdown() Implements:

$$\text{shutdown} = \begin{cases} \text{True} & \text{if } V < V_{\text{shutdown}} \\ \text{False} & \text{otherwise} \end{cases}$$

```python
def _check_shutdown(self, voltage: float) -> bool:
    return voltage < self.config.v_shutdown
```

Input: `voltage` (V)
Output: `shutdown` (bool)

4.4 4.5 Main Simulation Loop

Function: step() Execution sequence:

1. Harvest: $P_{\text{harv}} = \eta_{\text{pv}} A_{\text{pv}} G(t)$
2. Select: `action = controller(V, B)`
3. Load: $E_{\text{load}} = V \times I \times T$
4. Update energy: $E'_{\text{cap}} = E_{\text{cap}} + E_{\text{harv}} - E_{\text{load}}$
5. Update voltage: $V' = \sqrt{2E'_{\text{cap}}/C}$
6. Update buffer: $B' = B + r_{\text{in}} \times T - n_{\text{proc}}$
7. Check shutdown: `shutdown = (V' < V_shutdown)`
8. Log state
9. Advance time: $t' = t + \Delta t$

---

5 Extension Points for RL Integration

When integrating reinforcement learning (Weeks 4–6), we will replace:

Current rule-based controller:

```
def _select_action(self):
    # Rule-based logic
    if v >= threshold_high:
        return HIGH
    # ...
```

With RL agent:

```
def _select_action(self):
    # RL-based decision
    state_vector = self._get_state_features()
    q_values = self.q_table.get_q_values(state_vector)
    return argmax(q_values)
```

State features for RL (So far):

- Current voltage (discretized)
- Buffer occupancy (discretized)
- Recent irradiance trend
- Time of day
- Energy gradient (charging/discharging)

---

6 Model Assumptions and Limitations

6.1 Energy Harvesting Assumptions

Assumed:

1. **Solar panel operates at Maximum Power Point (MPP)**: η_{pv} constant. Impact: may overestimate at low irradiance.
2. **Uniform irradiance across panel**: no partial shading or soiling.
3. **PMIC efficiency is constant**: $\eta_{\text{PMIC}} = 85\%$ independent of load.
4. **No temperature effects**: panel efficiency independent of temperature.
5. **Instantaneous power transfer**: no MPPT tracking time or inrush currents.

Not Currently Modeled:

- PMIC quiescent current (10–100 μA)
- MPPT algorithm power consumption
- Reverse leakage during night
- Panel aging/degradation
- Spectral response (indoor vs outdoor)

6.2 Supercapacitor Assumptions

Assumed:

1. Ideal capacitor behavior: $E = \frac{1}{2}CV^2$, constant C
2. No self-discharge
3. Zero ESR (no I^2R losses)
4. Instantaneous voltage response
5. No voltage sag during load

Not Currently Modeled:

- Leakage (self-discharge)
- ESR losses
- Voltage-dependent capacitance
- Temperature effects on C and ESR
- Aging and cycle life

6.3 Load/MCU Power Assumptions

Assumed:

1. Constant current during execution (I_high = 8.0 mA)
2. Fixed execution times
3. Instantaneous mode transitions (no wake-up cost)
4. Sleep current includes all idle components (50 μA)
5. No baseline/quiescent board power during active modes

Not Currently Modeled:

- Voltage regulator quiescent current (10–50 μA)
- Memory refresh power
- Peripheral standby power
- Flash access power
- Oscillator power
- Wake-up transition energy
- Temperature-dependent MCU power

6.4 Buffer and Data Flow Assumptions

Assumed:

1. Continuous input stream at $r_{\text{in}} = 2$ fps
2. Instantaneous processing
3. Lossless buffer (drops when full)
4. No separate memory access power
5. Fractional frames allowed in simulation (for convenience)

Not Currently Modeled:

- DMA power
- Buffer memory retention power
- IPC overhead
- Frame metadata storage

6.5 Control and Decision Logic Assumptions

Assumed:

1. Perfect, instantaneous voltage sensing (no noise)
2. Zero-overhead decision making (no CPU energy)
3. Hard voltage thresholds (no hysteresis)
4. No prediction/forecasting included
5. Deterministic behavior (no randomness)

Not Currently Modeled:

- Hysteresis
- Voltage filtering/moving average
- Predictive control
- Multi-step lookahead
- Safety margins and conservative operation
- Checkpoint/restore overhead

6.6 8.7 Simulation Framework Assumptions

Assumed:

1. Fixed timestep $\Delta t = 0.1$ s
2. Synchronous update sequence each timestep
3. Simple Euler integration (no advanced integrators)
4. Floating-point arithmetic (infinite precision assumption)

Not Currently Modeled:

- Interrupt handling, RTOS overhead, communication overhead
- Sensor synchronization timing
- Watchdog behavior, error handling

6.7 Missing Physical Phenomena

Not modeled items include:

- Thermal effects (panel, MCU, capacitor)
- Aging and degradation (panel, capacitors)
- Electrical parasitics (wires, inductance)
- Communication and I/O power (radio, sensors)
- Safety and reliability mechanisms