Solar + HHO Hybrid Pipeline Energy System

Integrating photovoltaic generation, HHO electrolysis, and fuel cell storage in a unified Lateralus control pipeline

Lateralus Language

bad-antics · April 2026 · Off-Grid Systems Research

ABSTRACT This paper describes the engineering design and control software for a hybrid off-grid energy system that combines photovoltaic solar panels, HHO electrolysis for medium-term energy storage, and a PEM fuel cell for on-demand generation. The system is controlled by a Lateralus pipeline program that reads sensor telemetry, computes a dispatch plan, and issues control commands to each subsystem. A formal energy balance model is included. The system has been simulated in QEMU against synthetic solar irradiance data and validated at bench scale (1 kW).

1. System Overview

The hybrid system has three energy pathways:

```
Solar panels (1-50 kW)
    ↓ MPPT charge controller
    ■■■→ AC loads (via inverter)          [direct path]
    ■■■→ Battery bank (0.5-10 kWh)        [short-term storage]
    ■■■→ Electrolyzer (surplus)            [medium-term storage]

Electrolyzer
    ↓ produces H2 + O2
    ■■■→ Compressed gas tanks (50-500 kWh)

Gas tanks
    ↓ when needed
    ■■■→ PEM fuel cell → battery → loads  [discharge path]
```

The battery bank serves as a buffer between the millisecond-scale solar variability and the minute-scale response of the electrolyzer and fuel cell. The HHO system handles day-to-week timescale storage.

2. Solar Generation Model

Solar generation is modeled using a simplified one-diode photovoltaic model parameterized by irradiance G (W/m²) and cell temperature T_c (°C):

```
-- Simplified PV power output
fn pv_power(G: f32, T_c: f32, panel: PanelSpec) -> Watts {
    let eta_temp = 1.0 - panel.temp_coeff * (T_c - 25.0);
    let p_stc    = G / 1000.0 * panel.rated_w;
    Watts(p_stc * eta_temp * panel.efficiency)
}

-- Typical parameters (monocrystalline, 400W panel):
--   rated_w = 400
--   temp_coeff = 0.0035 /°C
--   efficiency = 0.21
```

The MPPT controller tracks the maximum power point using the perturb-and-observe algorithm, updated every 100ms. The controller output is a DC bus voltage command to the boost converter.

3. Electrolyzer Control

The electrolyzer is activated when surplus solar power exceeds a minimum threshold (to avoid inefficient partial-load operation) and the H2 tanks are not full:

```
fn electrolyzer_setpoint(
```

```
    surplus: Watts,
    tank_pressure: Bar,
    config: &ElectrolyzerConfig,
) -> ElectrolyzerCommand {
    if surplus < config.min_power || tank_pressure >= config.max_bar {
        ElectrolyzerCommand::Off
    } else {
        let clamped = surplus.clamp(config.min_power, config.max_power);
        ElectrolyzerCommand::SetPower(clamped)
    }
}
```

The electrolyzer takes 2-5 minutes to warm up from cold start. The controller hysteresis prevents rapid cycling: the electrolyzer stays on for at least 30 minutes after activation and stays off for at least 15 minutes after shutdown.

4. Fuel Cell Dispatch

The fuel cell is activated when solar generation plus battery discharge cannot meet load demand. The dispatch logic uses a state machine with three states:

```
enum FuelCellState { Standby, WarmingUp, Running }

fn fc_dispatch(
    deficit: Watts,
    soc: Pct,
    state: FuelCellState,
) -> FuelCellCommand {
    match state {
        Standby    if deficit > 200.0 && soc < 30.0 => Start,
        WarmingUp  if warm_up_complete()             => Enable,
        Running    if deficit < 50.0 && soc > 70.0  => Shutdown,
        _                                            => Hold,
    }
}
```

The fuel cell warm-up period is 3-8 minutes (PEM, cold start). During warm-up, the battery bank supplies the deficit. If the battery reaches 10% SOC before warm-up completes, the fuel cell is forced to minimum power early (partial efficiency accepted).

5. The Control Pipeline

The system controller is a Lateralus pipeline running at 1 Hz:

```
fn control_loop(sys: &mut SystemState) -> Result<ControlPlan, Fault> {
    sys
        |>  read_all_sensors           // solar, battery, tanks, loads
        |?> validate_sensor_readings   // fault on out-of-range
        |>  compute_energy_balance     // surplus or deficit
        |>  update_electrolyzer_cmd    // write to electrolyzer
        |>  update_fc_state_machine    // write to fuel cell
        |>  update_inverter_cmd        // write to inverter
        |>  log_telemetry              // append to time series DB
}
```

Each stage reads from and writes to the SystemState record. The pipeline is synchronous: all stages complete before the next 1 Hz tick. Sensor reads are buffered asynchronously in hardware; the read_all_sensors stage only copies from the buffer.

6. Energy Balance Verification

The energy balance at each time step must satisfy:

```
-- Conservation equation (Watts)
P_solar + P_fuel_cell = P_load + P_electrolyzer + P_battery_charge
  (where P_battery_charge is negative if discharging)

-- The controller verifies this after each dispatch
fn verify_balance(plan: &ControlPlan) -> Result<(), EnergyFault> {
    let lhs = plan.solar + plan.fuel_cell;
    let rhs = plan.load + plan.electrolyzer + plan.battery_delta;
    if (lhs - rhs).abs() > BALANCE_TOLERANCE {
        Err(EnergyFault::ImbalancedPlan { lhs, rhs })
    } else { Ok(()) }
}
```

An energy imbalance fault triggers the safe shutdown sequence: all controllable loads are shed in priority order, then the fuel cell and electrolyzer are shut down. The fault is logged with the full sensor state for post-mortem analysis.

7. Simulation and Validation

The control software is validated using a hardware-in-the-loop simulator. The simulator provides a synthetic environment with configurable solar irradiance, load profiles, and fault injections:

```
# Run 7-day simulation with winter solstice irradiance
ltl run energy::simulator \
    --irradiance data/solstice_7day.csv \
    --load data/residential_load.csv   \
    --config examples/10kw-system.toml \
    --report sim_results.json

# Results (7-day, December solstice, 52°N latitude):
#   Total solar generation:    312 kWh
#   Load served:              210 kWh
#   Electrolysis:              89 kWh (stored as H2)
#   Fuel cell generation:      45 kWh
#   Energy balance error:     < 0.1% (rounding only)
```

8. Future Work

The current system design has three open engineering problems:

- **Predictive dispatch**: the current controller is reactive. A model-predictive controller using weather forecast data would pre-activate the electrolyzer before predicted surplus and pre-warm the fuel cell before predicted deficit.
- **Multi-site coordination**: the LEPP protocol (companion paper) enables multiple sites to share surplus energy via mesh networking. The dispatch algorithm needs extension to handle remote energy flows.
- **Formal verification**: the energy balance conservation law should be verified as an invariant of the control pipeline using the Lateralus formal verification framework. This would eliminate the possibility of a dispatch plan that violates conservation.

Lateralus is an open-source, zero-dependency programming language. Project home: https://lateralus.dev. Source: github.com/bad-antics/lateralus-lang. Released under CC BY 4.0.