Gaszähler

- über Hall Sensor
- https://www.laszloturi.eu/2023/12/how-to-integrate-mechanical-gas-meter.html
- SS49E.PDF
- https://arduinomodules.info/ky-024-linear-magnetic-hall-module/
- https://github.com/red5alex/hall-effect-gas-meter-sensor

Aufbau

- Sensor : SS49E

Kalibrieren

ALT

download

```
substitutions:
  devicename: "iot-kl-wl-sc-gaszaehler"

esphome:
  name: iot-kl-wl-sc-gaszaehler
  friendly_name: IOT_KL_WL_SC_GasZaehler

rp2040:
  board: rpipicow

preferences:
  flash_write_interval: 5min  # set to 5min to prevent wearing out the
onboard flash module too quickly

# Enable logging
logger:
  baud_rate: 115200
  level: DEBUG

status_led:
  pin: 32

globals:
  - id: total_pulses          # Gesamte Impulse (Counter)
    type: int
    restore_value: true       # Speichert im Flash (nach Neustart
```

```
erhalten)
    initial_value: '0'
  - id: impuls_factor         # Kalibrierung: m³ pro Impuls (z. B.
0.01)
    type: float
    restore_value: true
    initial_value: '0.01'     # Passe an deinen Zähler an!
  - id: initial_consumption   # Offset: Aktueller Zählerstand zum Start
    type: float
    restore_value: true
    initial_value: '1234.56'  # Dein aktueller Zählerstand (hier
kalibrieren!)

# Enable Home Assistant API
api:
  encryption:
    key: !secret api_encryption_key
  services:
      - service: set_initial_gas_reading
        variables:
          reading: float
        then:
          - lambda: |-
              id(initial_consumption) = reading;
              ESP_LOGI("calib", "Initialwert gesetzt: %.3f m³",
reading);
          - homeassistant.service:
              service: persistent_notification.create
              data:
                title: "Gaszähler kalibriert"
                message: "Neuer Startwert: {{ reading }} m³"

ota:
  - platform: esphome
    password: !secret ota_password

wifi:
  ssid: !secret wifi_ssid
  password: !secret wifi_password
  ap:
    ssid: "Gaszaehler Fallback Hotspot"
    password: "schmidt01"
  #manual_ip:
  #  static_ip: 192.168.30.133
  #  gateway: 192.168.30.1
  #  subnet: 255.255.255.0
binary_sensor:
  - platform: gpio
    pin:
      number: GPIO22  # Beliebiger freier GPIO (z. B. GP22 = Pin 29)
      mode: INPUT_PULLUP
```

```
      inverted: true
    name: "Gaszähler Impuls"
    id: gas_pulse
    filters:
      - delayed_on: 100ms  # Entprellung
    on_press:
      then:
        - lambda: |-
            id(total_pulses)++;
            ESP_LOGD("gas", "Impuls! Total: %d", id(total_pulses));
        - homeassistant.service:
            service: notify.mobile_app
            data:
              message: "Gaszähler: +{{ id(impuls_factor) }} m³"

sensor:
  - platform: uptime
    name: "RP2040 Uptime"
    unit_of_measurement: "s"
    icon: mdi:timer-outline
  - platform: internal_temperature
    name: "RP2040 CPU Temperatur"
    unit_of_measurement: "°C"
    device_class: temperature
    state_class: measurement
    icon: mdi:chip
# === SENSOR: Verbrauch in m³ ===
  - platform: template
    name: "Gasverbrauch m³"
    id: gas_m3
    device_class: gas
    state_class: total_increasing
    unit_of_measurement: "m³"
    accuracy_decimals: 3
    update_interval: 10s
    lambda: |-
      return id(initial_consumption) + (id(total_pulses) *
id(impuls_factor));
# === Optional: Energie in kWh (mit Brennwert) ===
  - platform: template
    name: "Gas Energie kWh"
    device_class: energy
    state_class: total_increasing
    unit_of_measurement: "kWh"
    accuracy_decimals: 2
    update_interval: 10s
    lambda: |-
      float brennwert = 11.2;     // Aus Gasrechnung (kWh/m³)
      float zustandszahl = 0.95;  // Aus Rechnung
      return (id(initial_consumption) + (id(total_pulses) *
id(impuls_factor))) * brennwert * zustandszahl;
```

Neu

download

```
substitutions:
  devicename: "iot-kl-wl-sc-gaszaehler"

esphome:
  name: iot-kl-wl-sc-gaszaehler
  friendly_name: IOT_KL_WL_SC_GasZaehler

esp32:
  board: esp32-c3-devkitm-1
  framework:
    type: esp-idf

preferences:
  flash_write_interval: 5min  # set to 5min to prevent wearing out the
onboard flash module too quickly

# Enable logging
logger:
  baud_rate: 115200
  level: DEBUG

# Enable Home Assistant API
api:
  encryption:
    key: !secret api_encryption_key
#  services:
#      - service: set_initial_gas_reading
#        variables:
#          reading: float
#        then:
#          - lambda: |-
#              id(initial_consumption) = reading;
#              ESP_LOGI("calib", "Initialwert gesetzt: %.3f m³",
reading);
#          - homeassistant.service:
#              service: persistent_notification.create
#              data:
#                title: "Gaszähler kalibriert"
#                message: "Neuer Startwert: {{ reading }} m³"

ota:
  - platform: esphome
    password: !secret ota_password

wifi:
  ssid: !secret wifi_ssid
```

```
  password: !secret wifi_password
  ap:
    ssid: "Gaszaehler Fallback Hotspot"
    password: "schmidt01"
  #manual_ip:
  #  static_ip: 192.168.30.133
  #  gateway: 192.168.30.1
  #  subnet: 255.255.255.0

# Enable Web server
web_server:
  port: 80
  version: 3

button:
  - platform: restart
    name: "Restart Device"
    id: restart_button
    icon: "mdi:restart"

# DEBUGGING (inkl. Mem Usage)
# https://esphome.io/components/debug/
debug:
  update_interval: 180s

status_led:
  pin: GPIO8

# Time component to sync with Home Assistant
time:
  - platform: homeassistant
    id: ha_time

# Define global variables for thresholds (adjust these as needed)
globals:
  - id: low_threshold
    type: float
    restore_value: no
    initial_value: '1.8'      # Example low threshold (normalized ADC
value, 0-1 for ESP32)
  - id: high_threshold
    type: float
    restore_value: no
    initial_value: '2.4'      # Example high threshold (normalized ADC
value, 0-1 for ESP32)
  - id: impulse_value
    type: float
    restore_value: no
    initial_value: '0.1'      # m³ per impulse - adjust based on your
meter (e.g., 0.01 for 100 impulses per m³)
```

```
# Persistent impulse counter
  - id: gas_total_impulses
    type: int
    restore_value: true
    initial_value: '0'

# Period tracking globals (last_period restore no to reinitialize on
boot)
  - id: last_hour
    type: int
    restore_value: no
    initial_value: '-1'
  - id: last_day
    type: int
    restore_value: no
    initial_value: '-1'
  - id: last_week
    type: int
    restore_value: no
    initial_value: '-1'
  - id: last_month
    type: int
    restore_value: no
    initial_value: '-1'
  - id: last_year
    type: int
    restore_value: no
    initial_value: '-1'

  # Start totals for each period (restore true to persist values)
  - id: hourly_start_total
    type: float
    restore_value: true
    initial_value: '0.0'
  - id: daily_start_total
    type: float
    restore_value: true
    initial_value: '0.0'
  - id: weekly_start_total
    type: float
    restore_value: true
    initial_value: '0.0'
  - id: monthly_start_total
    type: float
    restore_value: true
    initial_value: '0.0'
  - id: yearly_start_total
    type: float
    restore_value: true
    initial_value: '0.0'
```

```
# Editable number for initial consumption (persistent, adjustable in
HA)
number:
  - platform: template
    name: "Initial Gas Consumption"
    id: initial_consumption
    optimistic: true
    restore_value: true
    initial_value: 24000.56  # Setze hier deinen aktuellen Zählerstand
(persistent nach Änderung)
    min_value: 0
    max_value: 1000000
    step: 0.01
    unit_of_measurement: "m³"
    icon: "mdi:counter"
    mode: box

sensor:
  - platform: adc
    pin: GPIO3
    name: "Hall Sensor ADC"
    id: hall_adc
    update_interval: 100ms
    attenuation: auto  # Or 11db for full range
    unit_of_measurement: "V"
    accuracy_decimals: 2
    internal: True
    filters:
      - median:
          window_size: 5
          send_every: 3

  # Template sensor for total impulses (exposes the global)
  - platform: template
    name: "Total Gas Impulses"
    id: gas_total_impulses_sensor
    unit_of_measurement: "impulses"
    icon: "mdi:pulse"
    accuracy_decimals: 0
    update_interval: 60s
    lambda: |-
      return id(gas_total_impulses);

# Template sensor for total consumption
  - platform: template
    name: "Total Gas Consumption"
    id: total_gas_consumption
    unit_of_measurement: "m³"
    icon: "mdi:counter"
    accuracy_decimals: 2
    update_interval: 1s
```

```
    lambda: |-
      return id(initial_consumption).state + (id(gas_total_impulses) *
id(impulse_value));

# Period checker (runs every 60s to detect period changes and reset
starts)
  - platform: template
    name: "Period Checker"  # Hidden or remove name if not needed in HA
    id: period_checker
    update_interval: 60s
    lambda: |-
      auto now = id(ha_time).now();
      if (!now.is_valid()) return NAN;

      float current_total = id(total_gas_consumption).state;

      // Hourly
      int current_hour = now.hour;
      if (current_hour != id(last_hour)) {
        id(hourly_start_total) = current_total;
        id(last_hour) = current_hour;
      }

      // Daily
      int current_day = now.day_of_month;
      if (current_day != id(last_day)) {
        id(daily_start_total) = current_total;
        id(last_day) = current_day;
      }

      // Weekly (Monday-based week number)
      char week_buf[3];
      now.strftime(week_buf, sizeof(week_buf), "%W");
      int current_week = atoi(week_buf);
      if (current_week != id(last_week)) {
        id(weekly_start_total) = current_total;
        id(last_week) = current_week;
      }

      // Monthly
      int current_month = now.month;
      if (current_month != id(last_month)) {
        id(monthly_start_total) = current_total;
        id(last_month) = current_month;
      }

      // Yearly
      int current_year = now.year;
      if (current_year != id(last_year)) {
        id(yearly_start_total) = current_total;
        id(last_year) = current_year;
```

```
    }

    return 0.0;  // Dummy value, hide this sensor in HA if possible

# Periodic consumption sensors (update every 60s)
- platform: template
  name: "Gas Consumption Hourly"
  id: hourly_gas
  unit_of_measurement: "m³"
  icon: "mdi:clock-outline"
  accuracy_decimals: 2
  update_interval: 60s
  lambda: |-
    return id(total_gas_consumption).state - id(hourly_start_total);

- platform: template
  name: "Gas Consumption Daily"
  id: daily_gas
  unit_of_measurement: "m³"
  icon: "mdi:calendar-today"
  accuracy_decimals: 2
  update_interval: 60s
  lambda: |-
    return id(total_gas_consumption).state - id(daily_start_total);

- platform: template
  name: "Gas Consumption Weekly"
  id: weekly_gas
  unit_of_measurement: "m³"
  icon: "mdi:calendar-week"
  accuracy_decimals: 2
  update_interval: 60s
  lambda: |-
    return id(total_gas_consumption).state - id(weekly_start_total);

- platform: template
  name: "Gas Consumption Monthly"
  id: monthly_gas
  unit_of_measurement: "m³"
  icon: "mdi:calendar-month"
  accuracy_decimals: 2
  update_interval: 60s
  lambda: |-
    return id(total_gas_consumption).state - id(monthly_start_total);

- platform: template
  name: "Gas Consumption Yearly"
  id: yearly_gas
  unit_of_measurement: "m³"
  icon: "mdi:calendar"
  accuracy_decimals: 2
```

```
    update_interval: 60s
    lambda: |-
      return id(total_gas_consumption).state - id(yearly_start_total);

##### ESP Werte
  - platform: wifi_signal
    name: "ESP32 WiFi Signal"
    id: esp32_wifi_signal
    update_interval: 120s
  - platform: uptime
    name: "ESP32 Uptime (s)"
    unit_of_measurement: "s"
    update_interval: 120s
    icon: mdi:timer-outline
  - platform: internal_temperature
    name: "ESP32 CPU Temperatur"
    unit_of_measurement: "°C"
    update_interval: 120s
    device_class: temperature
    state_class: measurement
    icon: mdi:chip

##### DEBUGGING
  - platform: debug
    free:
      name: "Heap Free"
    #fragmentation:
    #  name: "Heap Fragmentation"
    block:
      name: "Heap Max Block"
    loop_time:
      name: "Loop Time"
    #psram:
    #  name: "Free PSRAM"
    cpu_frequency:
      name: "CPU Frequency"

binary_sensor:
  - platform: template
    name: "Hall Impulse Detector"
    id: hall_detector
    device_class: motion
    #internal: True
    lambda: |-
      static bool last_state = false;
      float current_value = id(hall_adc).state;
      bool current_state = (current_value > id(high_threshold));
      bool impulse_detected = (!last_state && current_state);
      last_state = current_state;
      return impulse_detected;
    #update_interval: 100ms
```

```
    filters:
      - delayed_off: 50ms
    on_press:
      then:
        - lambda: |-
            id(gas_total_impulses) += 1;
id(gas_total_impulses_sensor).publish_state(id(gas_total_impulses));
id(total_gas_consumption).publish_state(id(initial_consumption).state +
(id(gas_total_impulses) * id(impulse_value)));  // Optional: Force
total update

text_sensor:
  - platform: uptime
    name: "ESP32 Uptime"
    format:
      separator: " "
      days:      "D"
      hours:     "h"
      minutes:   "m"

  - platform: wifi_info
    ip_address:
      name: "ESP32 IP Address"
    ssid:
      name: "ESP32 Connected SSID"
    bssid:
      name: "ESP32 Connected BSSID"
    mac_address:
      name: "ESP32 Mac Address"
    scan_results:
      name: "ESP32 Latest WiFi Scan"
    dns_address:
      name: "ESP32 DNS Address"
  # DEBUGGING
  - platform: debug
    device:
      name: "Device Info"
    reset_reason:
      name: "Reset Reason"
```