Embedded Project Report – Pico 2 W Sensor Dashboard

Your Name

Course: Hårdvarunära programmering 2 – del 2
Submission: 2025-10-10

Abstract

This project demonstrates a complete embedded software solution on the Raspberry Pi Pico 2 W using MicroPython. It integrates sensor reading, display handling, user input via a rotary encoder, and robust MQTT communication. The focus of this report is on the code structure, scheduling strategy, and communication logic.

1 Introduction

The task was to build a system that measures a physical quantity, provides a local menu system, and publishes data to an MQTT broker:contentReference[oaicite:0]index=0. The solution was implemented entirely in MicroPython, with emphasis on asynchronous programming and robust interrupt handling.

2 System Overview

The hardware consists of:

- Raspberry Pi Pico 2 W
- SSD1309 OLED (SPI)
- INA219 sensor (I^2C) for voltage/current
- EC11 rotary encoder with push button:contentReference[oaicite:1]index=1

The main focus, however, is the software implementation.

3 Code Structure

The software is divided into logical modules:

3.1 Measurement Task

Samples voltage and current every 100 ms and calculates derived values such as power, Wh, and mAh.

Listing 1: Measurement task

```
async def task_measure(ms=100):
    global v,i,p,tprev
    while True:
        now=time.ticks_ms()
        dt=max(0.001, time.ticks_diff(now,tprev)/1000.0)
        tprev=now
        v = ina.voltage()
        i = ina.current()
        p = v*i
        trip_update(v,i,dt)
        await aio.sleep_ms(ms)
```

3.2 OLED User Interface

Three modes: Live values, Trip statistics, and Menu navigation. The interface is updated asynchronously.

3.3 Rotary Encoder Handling

Implemented in raw mode without debounce. A single scheduled drain avoids "schedule queue full" errors while ensuring every edge is processed.

Listing 2: RotaryRaw ISR scheduling

```
def _try_schedule(self):
    if not self._scheduled:
        self._scheduled = True
        try:
            micropython.schedule(self._drain, 0)
        except RuntimeError:
            self._scheduled = False
```

3.4 MQTT Client

Implements auto-reconnect, last will, periodic ping, and publishes sensor data every 500 ms. Topics include voltage, current, power, Wh, mAh, elapsed time, and UI page state.

Listing 3: MQTT publish logic

```
if time.ticks_diff(now, last_pub) >= PUBLISH_MS:
    client.publish(TOPIC_V, b"%.3f" % v)
    client.publish(TOPIC_I, b"%.4f" % i)
    client.publish(TOPIC_P, b"%.3f" % (v*i))
    client.publish(TOPIC_WH, b"%.5f" % trip_wh())
    client.publish(TOPIC_MAH, b"%.1f" % trip_mah())
```

4 Results

- Measurements were updated at 0.5 s intervals on the broker.
- OLED menus and trip reset worked reliably.
- Rotary encoder events were stable despite the lack of debounce.
- The system recovered from Wi-Fi or MQTT disconnects automatically.

5 Conclusion

The project meets all requirements: real sensor measurement, robust MQTT publication, and a working local menu system. The main challenge was reliable encoder handling, which was solved using a raw interrupt approach with a scheduled drain mechanism. The code structure makes it easy to extend with additional sensors or menu options.