TDT4258 Lab 4

Low-Power Embedded Systems

Ambient Light Control 💡

👨‍🏫 Teaching Assistant: Callum Gran

Agenda

1. Lab overview
2. Assignment goal
3. Microchip Try platform
4. Preliminaries & setup
5. Reading the analog input
6. LED control
7. The tasks overview
8. Comparing results
9. Submission and leaderboard
10. Common pitfalls

Lab 4 Overview

- **Goal:** Control an LED based on ambient light
- **Focus:** Power-efficient programming
- **Platform:** AVR128DB48 microcontroller
- **Tool:** Microchip Try (remote hardware access)
- **Language:** C (XC8 compiler)
- **Main objective:**
  Turn **LED ON when dark**, **OFF when bright**, while minimizing **power consumption**

Assignment Goal

- Read **analog input** from the ambient light sensor
- Toggle the LED when light crosses a **threshold**
- Compare different implementations:
    1. Busy-waiting
    2. Polling with sleep
    3. Interrupt-driven
    4. Core-independent (event system)
- Aim for the **lowest power draw**

Power Consumption Targets

Task	Method	Target Power
3	Busy-waiting	≤ 1.3 mA
3	Polling	≤ 700 µA
4	Interrupt-driven	≤ 200 µA
5	Core-independent	≤ 150 µA

✅ To earn points, your implementation must perform **better** than these values.

🏆 **Top 4 leaderboard entries** get a surprise in the final lecture!

Microchip Try Platform

🧠 Overview

- Run your code on **real hardware** online
- Two modes:
  - **Sandbox:** test & debug (2 min sessions)
  - **Challenge:** verification + power measurement
- Upload a `.hex` file for testing

Sandbox Mode

- Run known **test sequence**:
    - 5 light ON/OFF cycles over 10 seconds
    - ON for 400 ms, OFF for 1600 ms
- Observe LED reaction
- View power consumption graph
- Adjust ambient light manually
- Access UART terminal

🕒 Each session: 2 minutes hardware access

Challenge Mode

- Random ON/OFF sequence
  - Total ON time: 2 s
  - Total OFF time: 8 s
- System measures **power consumption**
- If LED timing is correct → eligible for **leaderboard submission**
- You can retry as many times as you want

Preliminaries

Development options:

1. **MPLAB Extensions for VS Code** (recommended)
2. **MPLAB X IDE**
3. Custom environment (unsupported)

Required tools:

- MPLAB XC8 compiler
- MPLAB Extension Pack for VS Code
- AVR128DB48 device support

📹 YouTube setup guide:
Microchip MPLAB in VS Code

Building and Uploading

Generating a `.hex` file

Add this post-build step:

```
avr-objcopy -O ihex %{TargetPath} %{TargetDirRelative}/%{TargetName}.hex
```

✅ Build with:

- `Ctrl+Shift+B` → Full Build
- Output:

  ```
  out/<project-name>/default.hex
  ```

Upload `.hex` file to Microchip Try Sandbox or Challenge mode.

USART Setup (optional)

- Download provided `usart.h` and `usart.c`
- Include in your project:

```c
#include "usart.h"
USART3_Init();
USART3_SendChar('A');
```

- Useful for debugging via serial terminal
- Default clock: `F_CPU 4000000UL`

Step 1: Initialize the Analog Comparator (AC)

- Configure **PD2** as analog input
- Disable digital buffer and pull-up
- Set **positive input** to the light sensor
- Set **negative input** to DACREF (voltage reference)

```c
void AC_init() {
  PORTD.DIRCLR = PIN2_bm;
  PORTD.PIN2CTRL = PORT_ISC_INPUT_DISABLE_gc;
  // ...
}
```

📖 Datasheet: Chapter 32.3.1

Step 2: Voltage Reference

```c
void VREF_init(void) {
  VREF.ACREF = VREF_REFSEL_1V024_gc;
}
```

- Sets analog comparator reference to **1.024 V**
- Call this before initializing the comparator

📖 Datasheet: Chapter 21.3.1

Step 3: LED Setup

```
void LED_init() { PORTA.DIRSET = PIN2_bm; }

void set_LED_on()  { PORTA.OUTCLR = PIN2_bm; }  // Active low
void set_LED_off() { PORTA.OUTSET = PIN2_bm; }
```

💡 LED ON when dark, OFF when bright

Make sure to test using the AC status bit:

```
if (AC0.STATUS & AC_CMPSTATE_bm) { /* bright */ }
```

Task 3: Busy-Waiting

- Continuously check light sensor output
- LED ON when below threshold, OFF when above
- Structure:

```
while(1) {
  if (AC_above_threshold()) set_LED_off();
  else set_LED_on();
}
```

- ⚠️ High power usage (no sleep)

Task 4: Polling with Sleep Mode

- Periodically check the sensor
- Sleep between checks
- Use **Timer/Counter A (TCA)** for wake-up interrupts
- Example:

```
TCA0.SINGLE.PER = 20000;  // 10 ms
TCA0.SINGLE.INTCTRL = TCA_SINGLE_OVF_bm;
```

- Adjust `PER` for timing accuracy

✅ Significantly lower power draw

Task 5: Interrupt-Driven Approach

- AC triggers interrupt on threshold crossing
- Enable interrupt vector: `AC0_AC_vect`
- ISR handles LED state and puts MCU back to sleep
- No need for polling or timer interrupts

⚡ Expected consumption: < 200 μA

Task 6: Core-Independent Operation

- Use **Event System**
  - Event generator: **AC output**
  - Event user: **LED pin**
- LED toggles automatically — CPU stays asleep
- No interrupts, no while-loop activity

💤 Achieves ultra-low power (≈150 µA or less)

Compare Results

Approach	Description	Power	CPU Activity
Busy-wait	Constant checking	🔴 High	100% active
Polling	Periodic checks + sleep	🟠 Medium	Partial sleep
Interrupt	Reacts to events	🟢 Low	Mostly asleep
Event system	Fully autonomous	🟢 Ultra-low	Always asleep

📈 Compare and discuss results in Blackboard submission.

Submission Requirements

- Submit:
  - `main.c` (Task 5 implementation)
  - **AI statement**
  - **Blackboard form** answers for Tasks 3–5
- Program must:
  - Correctly toggle LED based on light
  - Use requested low-power technique
- **Deadline:** Friday, 7th Nov 17:00
- Suprise for best 4 leaderboard entries 🎉

Wrap-Up

✅ Test in **Sandbox** before Challenge
✅ Compare current draw between methods
✅ Comment your code
✅ Submit before the deadline

📌 Ask questions on Piazza or during lab sessions

Time for help :)