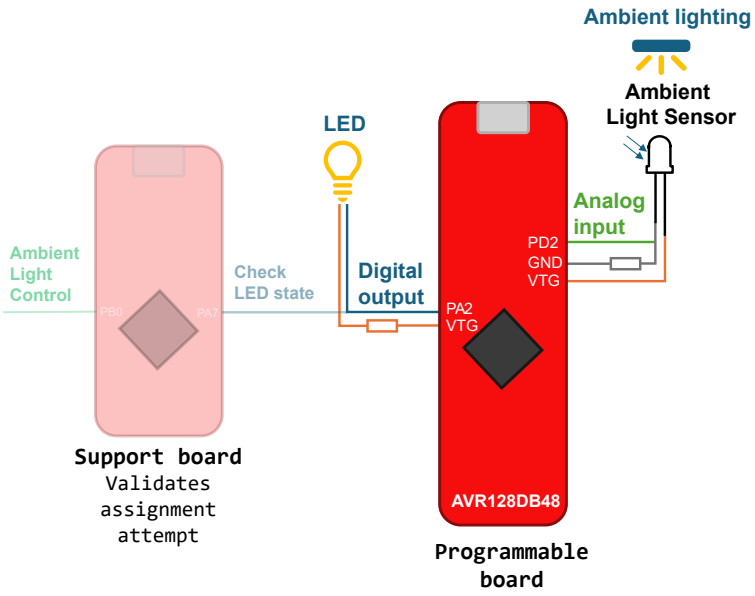
Assignment goal

In this lab assignment the goal is to create an application that reads the analog input from an analog ambient light sensor and uses the reading to turn off the LED when the ambient light level goes above a set threshold and turns it on when it goes below the threshold. The objective is to achieve this while keeping the power consumption as low as possible. That is, when it's dark: turn the LED on, when it's bright: turn the LED off.



The assignment consists of six tasks to guide you in building the application.

How to get points in Lab 4

In this lab you are trying to minimize energy consumption using different approaches. The different approaches result in vastly different power consumption. Tasks 3-5 implement those different approaches. To be awarded points for those tasks, your solution needs to show better (read: lower) power consumption than the indicated solutions on the leaderboard, which corresponds to the following limits:

- Task 3 busy-waiting: 1.3 mA,
- Task 3 polling: 700 µA,
- Task 4 interrupt-driven: 200 µA,
- Task 5 core independent operation: 150 µA.

The leaderboard numbers stem from a reference implementation, to which we added a bit of margin, such that you should be able to come up with a solution that outperforms the requirement for the respective tasks. For the hand-in on Blackboard, we ask you to submit your implementation of task 5. In addition to the file upload and the AI statement, you'll find some more text questions, where you can answer the questions raised in tasks 3-5.

Bonus

For the top four submissions on the leaderboard in lab 4 we have a small surprise in the last lecture!

About Microchip Try

Microchip Try allows you to remotely control hardware online. We'll be using the Microchip Try framework in this lab assignment. There are two modes available for this assignment: A *Sandbox* for

testing and debugging your code with 2 minutes of hardware access and a serial terminal, and *Challenge mode* for running tests on your code to verify it against the assignment goal.

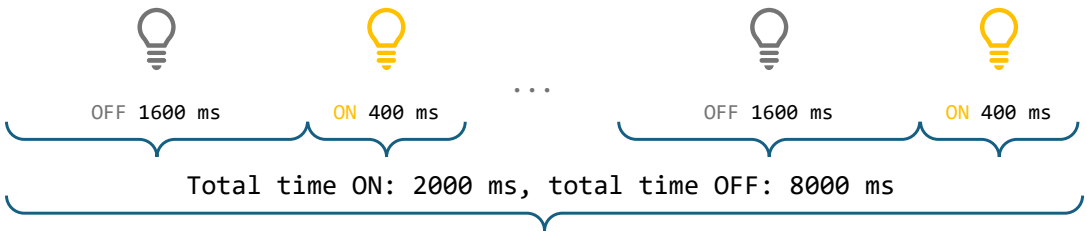
To program the microcontroller with your code, simply drag-and-drop a hex-file into the Microchip Try framework. See the *Preliminaries* tab on how to generate the hex-file.

🕒 Please note that you'll be running your code on physical hardware, and as such you may experience queueing and varying wait times depending on the amount of simultaneous users. If you upload a new hex-file while in queue, this will replace your previous file.

Sandbox

- Drag-and-drop programming
- Test your code
- Access to hardware 2 minutes at a time
- Access to serial terminal to communicate with the microcontroller
- Adjust the ambient light level manually
- Run a predefined, known test sequence, blinking the ambient lights on and off to test your implementation
- Get a measure of the power consumption used by the microcontroller

To check if your code achieves the assignment goal, you drag-and-drop your program (as a hex-file) in Try and then you can run a test sequence. The test sequence is illustrated below. The ambient lights go ON for 300 ms and then OFF for 1000 ms. This repeats five times for a total duration of 10 seconds.

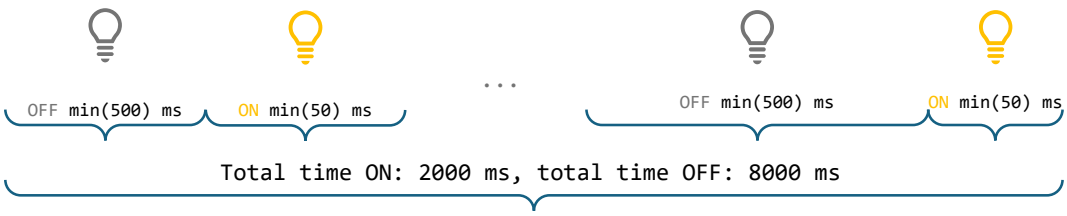


After the sequence has run, you get a response from Microchip Try letting you know whether your attempt was successful or not, i.e. whether you turned the LED on and off at the correct times according to the ambient light levels.

Challenge mode

In Challenge mode you can check if your code achieves the assignment goal, i.e. if your code is able to detect whether the ambient lights are ON/OFF and turn the LED OFF/ON accordingly. The power consumption of the microcontroller is measured while your code runs and the goal is to minimize the consumption.

To check if your code achieves the assignment goal, you drag-and-drop your program (as a hex-file) in Try. Your .hex-file is put in a queue and when the hardware is available, the microcontroller is programmed with your implementation. Then, a randomized sequence is generated, illustrated below. The ambient lights go ON for a minimum of 50 ms and then OFF for a minimum of 500 ms. This repeats five times for a total duration of 10 seconds.



After the sequence has run, you get a response from Microchip Try letting you know whether your attempt was successful or not. If you successfully turned the LED on and off at the correct times, you get the option to submit your attempt to the leaderboard.

You can try as many times as you wish.

Note on variations between hardware

Microcontrollers can have varying power consumption due to differences in their design and manufacturing processes. For the leaderboard, all participants will use the same hardware instance to enable more fair performance comparisons, although this may also result in some minor variations. The hardware used in the sandbox and the hardware used in the Challenge section are two different instances of the same setup. Consequently, you may observe variations in power consumption and performance when running the same code on different hardware instances.

Preliminaries

For developing the code, there are four options:

- **Using MPLAB X IDE**: A standalone development environment for Microchip microcontrollers.
- **Using MPLAB Xpress**: A simplified, cloud-based version of the standalone MPLAB X IDE. No download is required. User creation is required.
- **Using MPLAB Extensions for VS Code**: This is a new project from Microchip and is still in early beta. This means that not all features are supported yet.
- **Setting up your own environment**: There are alternatives to the options described above, but these are not covered here.

Using MPLAB X IDE

How to download the IDE and create a project

1. Download the MPLAB X IDE. For Linux and Mac, follow the installation walkthrough provided in this walkthrough for MPLAB X.
2. Open the downloaded installer.
3. Follow the prompts until the "Select Application" page. Here you only need to select `MPLAB X IDE (Integrated Development Environment)` and `8-bit MCUs`.
4. When the install is complete, check the box `XC compiler are not installed with the IDE [...]` or go directly to the download page for the XC8 compiler.
5. Download the MPLAB XC8 C-Compiler for your Operating System.
6. Open the installer and follow the prompts, or use this installation guide.

7. Open MPLAB X and follow these step-by-step instructions on how to create a new project. Select `AVR128DB48` as device and `XC8` as compiler.

How to create and add files to your project

Use these instructions to learn how to create and add files to your project.

How to build the project and generate the hex-file

To upload your code to Microchip Try, you'll need a hex-file of your program.

1. Follow the instructions at Microchip Developer Help on how to generate a production hex-file.
2. Open your MPLAB X project folder in File Explorer, likely at `C:\Users\your-username\MPLABXProjects\<project-folder>` (for Windows users).
3. Locate your .hex-file in the folder `<project-folder>\dist\default\production\`.

Using MPLAB Xpress

1. Open MPLAB Xpress.
2. On your first visit, you'll be prompted to create an account or log in.
3. Follow these step-by-step instructions on how to create a new project. Select `AVR128DB48` as the device.

⚠ **NOTE**: Your created project will exist in the cloud sandbox for **3 days** from your last activity. Be sure to save your work by either using `File > Export` to download your project or `View > Git` Controls to commit your project to a git repository.

🛠 **Disclaimer**: While MPLAB Xpress offers a convenient, cloud-based development environment, please be aware that it may occasionally exhibit instability or bugs. If you encounter any issues, consider using one of the alternative development options listed here.

How to create and add files to your project

To create a new file:

1. Click `File` and select `New File`.
2. Select the file type you wish to create.
3. Click `Next`.
4. Name your file and press `Finish`.

To add existing files to your project:

1. Right-click on the folder `Header Files` or `Source Files` and press `Add Existing Item...` to add .h- and .c-files, respectively.
2. A window should pop up. Drag files from your local File Explorer into the window in Xpress.
3. Select the file you wish to add and press `Open`.

How to build the project and generate the hex-file

To upload your code to Microchip Try, you'll need a hex-file of your program.

1. Follow the instructions at Microchip Developer Help on how to generate the hex-file.
2. A window should pop up prompting you to save your hex-file.

Using MPLAB Extensions for VS Code

Please note that this is a new project from Microchip and is still in early beta. This means that not all features are supported yet and the setup requires some additional steps in this early phase.

1. Open VS Code.
2. Press Ctrl+Shift+X, search for MPLAB and download "MPLAB Extensions for Visual Studio Code Extension Pack".
3. Go to the XC compiler download page and download the MPLAB XC8 C-Compiler for your Operating System. See the installation guide.
4. Open the installer and follow the prompts. ⚠ **Note**: Do not change the Installation Directory.
5. In VS Code, press Ctrl+Shift+P and type `MPLAB: Create new project`. Select `AVR128DB48` as device and `XC8` as compiler.

How to create and add files to your project

- Create new files by right-clicking in the Project Folder in the Explorer view (press Ctrl+Shift+E to open) and selecting "New File".
- To add existing files to your project, drag-and-drop the file into the Project Folder in the Explorer view. A pop-up will appear in the lower right corner asking to add the new source file to the MPLAB project - press "Yes" for .c-files. To ensure the files were added to the project, open the `<project-name>.mplab.json` file in the `.vscode` folder. Check that the .c-files are listed under `"fileSets"`:

```
"fileSets": [
        {
            "name": "default",
            "files": [
                "main.c", // Note that the files are comma-separated
                "usart.c"
            ]
        }
    ]
```

How to build the project and generate the hex-file

1. Create a `tasks.json` file by pressing Ctrl+Shift+P and type `Tasks: Configure Default Build Task` and select the option `<project-name>: default - Full Build`.
2. Remove the line `"isDefault":true` and the preceeding comma in the above line.
3. Download `tasks.json`. Open the file and copy the second task with the label `"Generate hexfile"` into your `tasks.json` file. Remember to comma-separate the tasks:

```
{
  "version": "2.0.0",
  "tasks": [
    {
      "type": "MPLAB-CMake"
      // ...
    }, // Note that the tasks are comma-separated
    {
      "label": "Generate hexfile"
      // ...
    }
  ]
}
```

4. Update all instances of "project-name" with your project name. You may also need to update the file path of the XC8 compiler in line 22: `"command": "C:\\'Program Files'\\Microchip\\xc8\\...`, depending on your system environment and XC8 version.
5. Now you can build your project by pressing Ctrl+Shift+B and selecting either `project-name: default - Full Build` or `Generate hexfile`. The latter option both builds the project and then generates a hex-file. If you wish to do a clean build, press Ctrl+Shift+P and type `MPLAB CMake: Clean Build`.

If you encounter issues generating the hex-file, try the following suggestions:

- Check the path format in the `tasks.json` file for the `"Generate hexfile"` task. Depending on your system environment, you might have to use forwardslashes `/`, instead of the double backslashes `\\`.
- Check the path to the XC8 compiler is correct in the `tasks.json` file for the `"Generate hexfile"` task.
- Have you replaced all instances of `project-name` in the `tasks.json` file with your actual project name?
- The variable `${workspaceFolder}` may cause issues. Try searching online for solutions or use the full path instead.
- Do a clean build by pressing Ctrl+Shift+P and type `MPLAB CMake: Clean Build`.

If you experience conflicts with other extensions, try creating a new profile in VS Code for the MPLAB Extensions.

Limitations of MPLAB Extensions for VS Code

- The Language Server Protocol (LSP) is not yet implemented for 8-bit microcontroller projects, meaning features such as code completion and syntax highlighting are not fully supported.
- You will most likely encounter some bugs. Please report any bugs or suggestions for enhancements using this feedback form.

Set up USART

Download `usart.h` and `usart.c` from this task description. The files implement a USART driver which may be useful while developing your code using the Microchip Try framework.

Add the files to your project. Refer to the *Preliminaries* tab on the left on how to do this in your development environment.

Modify main.c

Modify your main.c file to include the usart.h file, by adding the following line at the top:

```
#include "usart.h"
```

In the `main()` function, test the driver by calling `USART3_Init()` to initialize the USART peripheral and `USART3_SendChar('A')` to send a character.

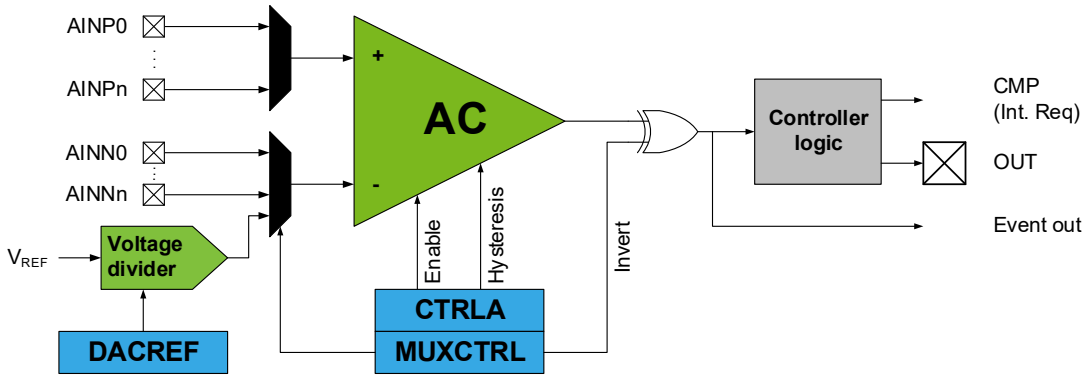
If you wish to, you can include the `<util/delay.h>` library in your main.c-file and use `_delay_ms()` to send a UART message every 1 second.

Note: the default clock frequency of the microcontroller is 4 MHz. This value is defined by `#define F_CPU 4000000UL` in `usart.h` and is used by the `<util/delay.h>` library and to calculate the baud rate of the USART communication. If you change the clock frequency of the microcontroller, remember to update the `F_CPU` value.

Generate the hex-file and upload your program to the sandbox in Microchip Try to run your code on the Curiosity Nano hardware.

Initialize the Analog Comparator

Read chapters 32.1 through 32.3 of the datasheet to get an overview of the Analog Comparator. Then, follow the initialization steps in chapter 32.3.1 to set up the peripheral, AC0. You'll need the table in chapter 3.1 to identify the correct positive input signal, `AINPn`, for pin PD2. The negative input signal should be set to 'DACREF'.



Step one, configuring the desired pin as an analog input, is provided below for the pin connected to the ambient light sensor:

```
void AC_init(){
    // Set pin PD2 (port D, pin 2) as an input
    PORTD.DIRCLR = PIN2_bm;
    // Disable digital input buffer and pull-up resistor for pin PD2
    PORTD.PIN2CTRL = PORT_ISC_INPUT_DISABLE_gc;
    // Remaining initialization steps...
}
```

The next steps in the initialization use the registers of the Analog Comparator. If you find these steps challenging, a pseudocode project with comments to guide you are provided.

📖 Reading the attached link to *Writing C-code for AVR MCUs* is highly recommended to get you started.

Setting voltage reference

To set the voltage reference of the analog comparator, add the following function to your project:

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

The function sets the voltage reference of the analog comparator (AC) to 1.024 V, as described in Chapter 21.3.1 of the datasheet which explains the initialization steps of the Voltage Reference (VREF) peripheral.

Remember to call this function in your `main()` function.

Set the threshold value for AC to activate

The analog comparator compares two analog input voltages and outputs a signal level indicating when one of the inputs is higher than the other.

The hardware for this assignment is set up such that the positive input of the AC is connected to the ambient light sensor on pin PD2. For the negative input we'll use an internal reference (DACREF). The internal reference value is determined by the threshold for what is considered 'on' and 'off' for the ambient light levels. In this assignment, we'll define $0.1\mathrm{V}$ as this threshold. Meaning, when the analog signal from the ambient light level sensor is below $0.1\mathrm{V}$, the AC output is low/'0' and the LED should be ON, and the LED should be OFF when the analog signal is above $0.1\mathrm{V}$ and the AC output is high/'1'.

Use the equation below from 32.3.2.2 in the datasheet to calculate the DACREF value and add the following line to your `AC_init()` function: `AC0.DACREF = <calculated_DACREF>`. V_{REF} is the voltage reference of the AC set in the previous task, V_{DACREF} is the voltage level threshold discussed above, and the DACREF value should be a number between 0-255.

$$V_{DACREF} = \frac{DACREF}{256} \cdot V_{REF}$$

Reading the status of the Analog Comparator

Implement a function that checks if the signal from the analog sensor is above the set threshold. See Chapter 32.5.6 of the datasheet. You'll need to include the `<stdbool.h>` header file to use boolean types.

Hint: Check the `CMPSTATE` bit field of the `STATUS` register

```
if(AC0.STATUS & AC_CMPSTATE_bm)
```

Setting up LED

Paste the following functions that turn and off the LED connected to pin PA2 into your main file.

```
void LED_init() {
    PORTA.DIRSET = PIN2_bm;
}
void set_LED_on(){
    // LED is active low. Set pin LOW to turn LED on
    PORTA.OUTCLR = PIN2_bm;
}
void set_LED_off(){
    // LED is active low. Set pin HIGH to turn LED off
    PORTA.OUTSET = PIN2_bm;
}
```

Remember to call the `LED_init()` function in your `main()`-function. You may remove the `USART3_Init()` function and other function calls to the USART peripheral if you don't use these for testing and debugging. If you remove the `#include "usart.h"` line, make sure that any necessary includes from `usart.h` are added to your main file instead.

Test turning ON and OFF the LED with the function you implemented in the previous task: When the signal from the analog sensor is above the set threshold turn the LED off, and turn the LED on when the signal is below the threshold.

Generate the hex-file and upload your program to the sandbox in Microchip Try.

Busy-waiting

Implement a busy-waiting scheme continuously checking if the ambient light sensor data is above the set threshold. When the sensor data is above the threshold (i.e. the sensor data indicates a low light

environment) turn the LED ON. Otherwise, the LED should be OFF. Refer to the lecture slides on the busy-waiting scheme.

The structure of your main file should look something like this:

```
// #include ...
// #define ...

void AC_init() {
    // ...
}
void VREF_init() {
    // ...
}

void LED_init() {
    // ...
}

void set_LED_on() {
    // ...
}

void set_LED_off() {
    // ...
}

bool AC_above_threshold() {
    // Check the output of the Analog Comparator
}

int main() {
    AC_init();
    VREF_init();
    LED_init();

    while(1) {
        // Implement the busy-waiting scheme
    }

    return 0;
}
```

💡 Ensure unused digital I/O pins are not left floating. Enable internal pull-up resistors and disable the digital input buffer to minimize leakage and reduce power consumption. Refer to Chapters 18.3.2.3 and 18.3.2.4 in the datasheet for Pin Configuration and Multi-Pin Configuration.

Generate the hex-file and upload your program to the Challenge section in Microchip Try. Save the power consumption graph.

Polling

Implement a polling scheme that periodically checks if the ambient light sensor data is above the set threshold. Put the device in sleep mode between the periodic checks. A timer driver using Timer/Counter A (TCA) is implemented for you with an interrupt service routine. The functions needed to put the device to sleep are also provided.

The timer wakes the device every 10 ms = 0.01 s. You can adjust this by updating the value of the `TCA0.SINGLE.PER` register according to the following formula:

$$period;[s] = PER \cdot \frac{DIVn}{f_{clock}} = PER \cdot \frac{2}{4000000}$$

The `PER` value is a 16-bit value and can set to any value from 0 to 2^{16}. The prescaler is currently set to 2, but it can be adjusted to any of the values specified in Chapter 23.5.1.

Note that this formula assumes the clock frequency of the microcontroller is 4 MHz, if you have changed this, you'll need to update the formula accordingly.

Generate the hex-file and upload your program to the Challenge section in Microchip Try. Save the power consumption graph.

Compare results

Compare the power consumption and discuss some advantages and disadvantages of the two methods. Please write your answer in the Submission Form on Blackboard.

Interrupt-driven approach

Update your initialization function for the analog comparator to enable interrupts when the analog sensor input goes above or below the threshold value. Make sure the AC is set to operate in standby mode. Then, implement the Interrupt Service Routine (ISR) for the AC. Use the ISR from the previous task as a guide. The interrupt vector for the comparator is `AC0_AC_vect` and remember to clear the interrupt flag `AC_CMPIF_bm`. Since the analog comparator (AC) will now generate interrupts, you can remove the timer-interrupt from the polling scheme as it is no longer necessary.

See Chapter 32.3.4 about the available AC interrupt.

The LED should have the same behavior as in the previous task and the microcontroller should be set to sleep after the ISR runs.

Generate the hex-file and upload your program to the Challenge section in Microchip Try. Save the power consumption graph.

Compare results

Compare the power consumption with the previous tasks and discuss some advantages and disadvantages of using the interrupt-driven approach. Please write your answer in the Submission Form on Blackboard.

Core independent operation

Update your implementation to fulfill the requirements of the application without any intervention from the CPU by using the Event System. Ensure that the infinite while loop in your `main()` function is empty and that the solution does not use interrupts.

Refer to chapter 16 of the datasheet about the Event System. The Event Generator is the analog comparator and the event user is the pin connected to the LED. Remember to put the device to sleep.

Generate the hex-file and upload your program to the Challenge section in Microchip Try. Save the power consumption graph.

Compare results

Compare the results with the previous three tasks. Discuss situations where using the Event System would be more advantageous than an interrupt-driven approach, and vice versa. Please write your answer in the Submission Form on Blackboard.

(Optional) Lowering power consumption

The final task is to lower the power consumption as much as possible. Use the methods from the lecture on low power techniques and the datasheet to attempt to lower the power consumption even further.

If you wish to submit your attempt to the leaderboard, use the Challenge section in Microchip Try.

We'd really like your feedback!

If you have any feedback to the Microchip Try system, the assignment, the toolchain or anything else, please submit your feedback.

Support

If you need technical support for the assignment, you may contact the team at Microchip through the email address posted on Blackboard. Please note that replies should not be expected outside working hours.