Software Development Practice 1

Instructor: RSP <rawat.s@eng.kmutnb.ac.th>

Getting Started with MicroPython for ESP32

Objectives

- Learn how to write MicroPython code for ESP32 microcontroller boards.

Expected Learning Outcomes

After completing all tasks, students will be able to:

- Write MicroPython code to utilize the hardware resources of the ESP32 microcontroller.
- Explain differences between Python code for general-purpose computers and microcontrollers.
- Test MicroPython code with a simulator and real hardware.

MicroPython-ESP32 Hands-on Lab Activities

Part 1: Experiments with Real Hardware

Hardware Preparation

1. Download the latest MicroPython firmware for ESP32 and flash the **ESP32 Wemos Lolin32 Lite board** with it.

2. Create a simple MicroPython script (main.py) for LED blinking.

- Upload main.py to the ESP32 board.
- Reset the board and observe the LED status.

3. Create a .mpy version of the LED blinking script (main.mpy) and upload it to the ESP32 board.

- Download the MicroPython source code from https://github.com/micropython/micropython.
- Build **mpy-cross** (on Ubuntu or WSL2 Ubuntu).
- Use **mpy-cross** to convert main.py into main.mpy.
- Install and use **mpremote** to upload main.mpy to the ESP32 board.
- Ensure you have removed main.py from the ESP32 flash filesystem.
- Reset the board and observe the LED status.

4. Clean up: remove both main.py and main.mpy from the ESP32 filesystem.

Instructions for WSL2 / Ubuntu

```
# Install git and build tools (gcc, make, etc.)
$ sudo apt install git build-essential

# Clone source code from the MicroPython repo
$ cd $HOME
$ git clone https://github.com/micropython/micropython.git

# Change directory to the mpy-cross compiler source
$ cd micropython/mpy-cross

# Build the mpy-cross compiler
$ make

# Go back to the user's home folder
$ cd $HOME

# Install mpremote tool for file transfer and control
$ pip install mpremote

# Upload main.py to ESP32 filesystem
$ mpremote connect /dev/ttyUSB0 cp main.py :/main.py

# Soft reset ESP32 to run main.py
$ mpremote connect /dev/ttyUSB0 reset

# Remove main.py from ESP32 filesystem
$ mpremote connect /dev/ttyUSB0 fs rm main.py

# List files on ESP32 filesystem (to confirm removal)
$ mpremote connect /dev/ttyUSB0 fs ls /

# Compile main.py to main.mpy for ESP32/S2/S3 (Xtensa arch)
$ micropython/mpy-cross/build/mpy-cross -march=xtensawin ./main.py

# Upload compiled main.mpy to ESP32
$ mpremote connect /dev/ttyUSB0 cp main.mpy :/main.mpy

# Soft reset ESP32 to run main.mpy
$ mpremote connect /dev/ttyUSB0 reset

# Run the main.mpy
$ mpremote exec "import main"
```

Note: To access USB devices from **WSL 2 Ubuntu on Windows 10/11**, you need to install and use **usbipd-win**. For detailed instructions, see Microsoft’s guide: https://learn.microsoft.com/en-us/windows/wsl/connect-usb

Task 1) I/O Follower

1.1) Write and test the MicroPython script (main.py) for the Wemos Lolin32 Lite board with the following functions.

- In the main loop, read the value from a push button (digital input, active-low, internal pull-up enabled) and write this value to an LED output.
- Choose appropriate GPIOs for the push-button input and the LED output.

1.2) Use a digital oscilloscope with two channels to measure the delay between the transition on the input signal and the corresponding transition on the output signal.

1.3) Capture waveforms for inclusion in the report.

1.4) Rewrite the MicroPython script to use an interrupt-based method. Test the script with real hardware, measure the signals, and compare the results with those from steps 1.1–1.2.

1.5) Compare the performance of two versions: **main.py** and **main.mpy**.

Task 2) Sine Wave Signal Generation

2.1) Write and test the MicroPython script for the Wemos Lolin32 Lite board with the following functions.

- Create an output signal with a sinusoidal waveform (with a DC offset). Its amplitude must be between 0 V and 3 V.
- Use a table of precomputed sine values (N = 100) to store samples of a full-cycle sine wave.
- Use the sample values in the table to generate the sine wave output using the DAC1 or DAC2 unit of the ESP32.
- Update the output with the next sample (in circular mode) using a timer every 1.0 ms or 0.1msec.

2.2) Revise the code to allow the user to adjust the amplitude of the output signal using the analog value read from a trimpot (voltage divider) circuit. The analog input voltage must be between 0 V and 3.3 V.

2.3) Use a digital oscilloscope to measure the output signal, determine the frequency and capture waveforms for inclusion in the report.

Note:

- ADC1 has 8 channels on GPIO pin numbers 32-39.
- ADC2 has 10 channels on GPIO pin numbers 0, 2, 4, 12-15, and 25-27.
- However, ADC2 is shared with the Wi-Fi module. It is generally recommended to use ADC1 to avoid conflicts.

Task 3) PWM Generation

3.1) Write and test a MicroPython script for the **Wemos Lolin32 Lite** board with the following functions:

- Create a **PWM (Pulse Width Modulation)** output signal with a frequency of **1kHz**.
- Adjust the duty cycle of the PWM signal using the analog value read from a trimpot (a voltage divider). The analog input voltage must be between 0V and 3.3V.

3.2) Use a digital oscilloscope to measure the output signal and capture waveforms with different duty cycles for inclusion in the report.

Part 2: Experiment with Wokwi Simulator

Simple Arcade Game Simulation with MicroPython

Summary: This project involves designing and implementing a simple arcade-style game using an ESP32 board programmed in MicroPython and simulated with Wokwi.

1. Create a virtual circuit with the following hardware components. Make sure that all wiring connections are correct.

- ESP32 or ESP32-S3 board
- Analog XY joystick (e.g. used to move objects)
- Push buttons as inputs (At least one button for "start" and optionally others for game actions like "fire", "jump", or "reset".)
- SPI TFT LCD display (e.g. with the ILI9341 driver) as output (note: There are some open source MicroPython libraries available for ILI9341 drivers.)

2. Design your own game with the following three game stages. The use of open source third-party libraries is allowed.

I) Startup Screen:

- Display a welcome message (e.g., "Press START to begin").
- Wait for the user to press the "Start" button before starting the game loop.

II) Gameplay:

- Read joystick values to control an on-screen character or object.
- Update object position based on joystick input (e.g., move a square or sprite).
- Optionally, include logic for scoring, object collisions, or a countdown timer.

III) End Screen:

- Triggered by game over conditions (e.g., timer ends or player loses).
- Display game summary:
  - "Game Over"
  - Author name, game version, and optionally score or play time
  - Wait for a button press to restart or exit

3. Record your gameplay as a video clip with a duration between 3 to 5 minutes.
