Name	SID	Checkoff

Objectives: Design app that combines analog and digital GPIO, interrupts and timers. We also familiarize ourselves with the oscilloscope.

The application combines the following elements:

a) Buzzer playing tune

b) LED with PWM dimming

c) Two-axis joystick and pushbutton serving the following functions:

- Pushbutton toggles between playing tune and synthesizer function
- Left-right controls the frequency of the synthesizer
- Up-down controls the duty cycle of the synthesizer

The key to getting all this working is to construct and test each part individually, and then gradually assemble them.

Although simple, the setup demonstrates the capabilities and may serve as a starting point for more sophisticated applications. After completing the lab, you are encouraged to improve the setup (e.g. adjust the range of the joystick control for better sensitivity) and try other ideas. Or hook up a speaker for real sound.

Parts and Tools

HUZZAH32 board, joystick, resistors (available in the lab), buzzer, LED, solderless prototyping board.

Prelab

Since different sections depend on reach other, read the entire document before starting work.

Get started early on the prelab to have sufficient time for resolving potential problems. Use Piazza, office hours, and discussions to get help and your questions answered so that you will come completely prepared.

LED Brightness

The goal is to gradually modulate the intensity of an LED from off to fully on over a period of 5 seconds. When the LED reaches full brightness, turn it off and start over. Use the built-in LED of the HUZZAH32 for the prelab and switch to an external (and brighter) LED in the lab with an approx. 500 Ω resistor in series (to limit the current and avoiding frying the ESP32). Draw the circuit diagram for the LED connection in the space below:

Note: LEDs are polarized. Current enters the terminal with the slightly longer lead and exits the shorter lead.

a) Configure the Pin to which the LED is connected as an open-drain output. Verify that you can turn the LED on and off.

b) Initialize PWM timer 0 for the led Pin with 500 Hz and 50 % duty cycle. Vary the duty cycle and verify that you can control the intensity between fully on and off.

c) Now configure timer 0 to call a function (e.g. `led_cb`) at a regular interval (determine the correct period to get a 5 second cycle). Each time `led_cb` is called, increase the PWM duty cycle for the LED by 1 (reset when 100 is reached). Suggestion: use a global variable `brightness` to keep track of the LED state.

   Note 1: import PWM from machine.

   Note 2: After setting up the timer, the program continues. If there is no more code to execute, microphython returns control to the repl. `led_cb` continues to be called at the period you specified. If you are executing the program with `run` (`shell49`), issue the `repl` command to see output from print statements you may be using for debugging.

   Note 3: Reset the ESP32 to stop the timer. This also frees up PWM channels—if you get a message that there are no more channels, reset the board before running the program.

   Note 4: Note that the ESP32 uses several kinds of timers: several timers to set the frequency of PWM outputs, several timers for executing code at periodic intervals, and the deepsleep timer. Probably a few other ones as well. For clarity they are all called timer!

Now you have configured the LED with PWM do to its "light show" without processor intervention: the ESP32 is available to do other things, e.g. play a tune.

Important: Use a timer, not a loop to control the LED brightness. Although a loop will work when the LED is the only part to be controller, the objective of the lab is to do several things simultaneously. If you use a loop in this part, you will have to rewrite your code later with a timer (replacing the loop) to complete the lab!

Tune

Here we configure a PWM channel to output the frequencies corresponding to a tune we want to play. A sample tune is reproduced at the end of this document. Search the internet for alternatives, or compose your own.

Configure an output as open-drain. Remember that the microcontroller ties open-drain outputs to GND when set to logic 0, and open (i.e. not connected to GND or V_{DD}) when set to 1. Connect the buzzer in a circuit from 3.3 V to a resistor (approx. 150 Ω), to the microcontroller output. If you do not have a 150 Ω resistor use the LED for testing and replace with the buzzer during the lab session. In the space below, draw the schematic showing the ESP32 pin, resistor, buzzer, and all other relevant terminals and connections. Remember that the buzzer is polarized.

a) Configure a PWM timer to control the pin the buzzer is connected to. Verify that you can control the buzzer frequency with PWM. Use a different PWM timer than for the LED (e.g. 1).

b) Use a for-loop to play a tune. Don't forget a sleep statement in the loop, or the tune will be very short!

c) Analogous to the LED, set up timer 1 to change the frequency of the buzzer output to the next note in your tune. Start over after reaching the end of the tune. Again, use a **timer**, not a loop to implement this part!

Run the LED and buzzer timers simultaneously. In more interesting applications, you may be controlling a robot, play safety warning sounds, and send measurement results to the cloud, all at the same time!

Button Interrupt

The goal is an interrupt driven program that counts the number of times the joystick button is pressed and switches between playing a tune and operating the synthesizer each time the button is pressed. The joystick button suffers from very bad bounce—use the tricks to debounce you learned in the lecture!

Since you will solder the joystick only in the lab, use a wire between the pin configured to read the button and ground "to simulate" its function. In the lab replace the wire with the button.

a) Choose a pin for reading the button state and configure as input with pull-up enabled.

b) In a for-loop, read and print the pin state. Verify that it changes as you press and release the button (insert/remove the wire).

c) Remove the for-loop and instead define an interrupt handler and attach it to the pin. Does pressing the button (inserting the wire) result in a falling or raising transition? Configure the trigger accordingly! Use a global `counter` to keep track of the number of button presses. Each time the button is pressed, print out the updated count.

d) Notice how the count changes by more than one each time the button is pressed. Follow the approach from the lecture to "de-bounce" the switch. Verify that the count increases by only one each time the button is pressed. Try different delays until you get the correct behavior.

ADC & DAC

Read the description of the joystick at https://www.adafruit.com/product/512.

The joystick consists of two potentiometers coupled to the x- and y-axes of the control button. The potentiometers are wired to the Vcc and GND terminals and the center pickoffs to Xout and Yout, respectively. Sel is tied to GND when the control button is pressed, and open otherwise.

In the lab we will be using the Joystick to control the synthesizer frequency and duty cycle. Connecting Vcc and GND to 3.3 V and GND of the HUZZAH32, Xout and Yout voltages will follow the joystick position. We will use the ESP32's ADC to read those voltages.

In the space below, draw a circuit diagram showing the joystick connected to the HUZZAH32 for reading out its position and button state.

Since we do not have the joystick available, we'll use the HUZZAH32's DAC to generate an analog output voltage and read it right back with an ADC for verification of our code (and the DAC and ADC).

a) Connect one of the DACs (e.g. DAC1) to an ADC input (e.g. ADC6) of the the HUZZAH32 board.

b) Configure the DAC and ADC (what's the best attenuation for reading the DAC output and for reading the joystick outputs, respectively?) and write a loop that iterates over all DAC values (0 ... 255) and reads resulting voltage with the ADC. Print both the DAC code and the ADC readout during each loop iteration. Use a delay (e.g. 100 ms in the loop to give the DAC some time to settle and the ADC to perform the conversion). Verify that the ADC readout increases monotonically with the DAC code.

c) Remove the wire from the DAC to the ADC. What happens? Why?

Oscilloscope

Familiarize yourself with the function and operation of oscilloscopes following the tutorial at

https://learn.sparkfun.com/tutorials/how-to-use-an-oscilloscope

(search the web for many other resources including videos). Make sure you understand the function of the trigger.

Write a program that configures two **different** PWM timers for 5 kHz and 8 kHz respectively, with 20 % and 60 % duty cycle. Have it ready in the lab!

Prelab Checkoff

At the start of the lab session, show the following results from the prelab to the instructor to get credit.

Task	Points	Checkoff
LED brightness demo and program	10	
Tune program	10	
Button interrupt count and code	10	
ADC/DAC demo and code	10	
Oscilloscope trigger and code	10	

Lab

Solder Joystick

Solder the joystick and header to its breakout board. Make sure that the joystick is oriented correctly and that the pins are aligned correctly before inserting it into the board. **Excessive force will bend the**

pins, preventing the joystick from working.

Use an Ohm-meter and check the resistance between

- `Vcc` and `GND`
- `Sel` and `GND` (low when button pressed, high otherwise)
- `Xout` and `GND` (varies as a function of joystick angle)
- `Yout` and `GND` (varies as a function of joystick angle)

After verifying correct operation, connect the joystick to the HUZZAH32 and verify the button interrupt and ADC readout of `Xout` and `Yout`.

Checkoff: ☐ (10 points)

Oscilloscope

Start up the oscilloscope, connect a probe to channel 1, enable the channel and set it to high sensitivity. Do not connect anything to the probe. Without touching anything, try to pick up the 60 Hz from the power lines.

This is an example of interference and of course affects not only oscilloscope probes, but all electronic circuits. Building circuits that are robust to such interference (i.e. their operation unaffected by it) is a quality of a good engineer!

Run the program from the prelab that sets up two PWM outputs to check out the oscilloscope. Connect an oscilloscope probe to one of the PWM outputs. Adjust the trigger such that the waveform is stable (i.e. does not run across the display). Set appropriate scaling to read the frequency and duty cycle off the display. Vary the duty cycle and verify that the oscilloscope output changes accordingly.

Now connect the second probe to display both PWM waveforms simultaneously. Choose to trigger on the first, then the second channel. Can you get both images to be stable, i.e. not run across the screen? Why not?

Checkoff: ☐ (10 points)

Synhesizer and LED Testing

Modify the callback handler playing the tune to have two states:

a) Play the tune, or

b) Operate the synthesizer.

In synthesizer mode, the buzzer plays a tone whose frequency and duty cycle are controlled by up-down and left-right motion of the joystick. In "play the tune" mode, the tune is played over and over.

The state changes (reliably, remember debouncing!) with each press of the joystick button.

If each part from prior section works as intended (in particular, you used timers and not loops to control the LED brightness and playing the tune), then this last part is little more than putting the code from each part into a single file and changing the code in the button interrupt handler to control the state (rather than count the number of presses).

Use your debugging skills acquired in prior labs. Make small incremental changes, and verify that each piece operates as required at each step. Go back and test individual parts if a complex setup is not working as intended. Check voltages (including the supplies for all components) with the DMM and waveforms with the oscilloscope. Use the continuity tester to verify connections (turn off power during this step to avoid confusing the meter).

Your program should be entirely interrupt and timer driven and return to the repl after everything is set up. Later we will add to the program to operate a robot while playing music.

If you do not finish during the lab you may complete your work later and submit for checkoff in the following lab session. Full credit for the immediately following lab session reduced/no credit later. Remember that you are required to complete all labs to pass the course.

Checkoff: ☐ (30 points)

Better Sound

The sound quality of the buzzer is pretty bad, if "sound" is even the right word. Easily fixed by replacing it with an audio amplifier and speaker. Many choices are available, e.g. https://www.adafruit.com/product/2130 or, for stereo and I2C volume control, https://www.adafruit.com/product/1712. Some speaker options: https://www.adafruit.com/product/1314 or https://www.adafruit.com/product/1313. These and similar parts are available from many vendors including major online sources.

Note that the speaker supply must be connected to a good 5 V source, e.g. VUSB. The 3.3 V output of the HUZZAH32 does not have sufficient current drive for this application.

Of course the ESP32 can synthesize several tones simultaneously and suitable circuits can combine the outputs for stunning effects. We will discuss circuits that accomplish this later in the course.

Sample Tune

```
# define frequency for each tone
C3  = 131
CS3 = 139
D3  = 147
DS3 = 156
E3  = 165
F3  = 175
FS3 = 185
G3  = 196
GS3 = 208
A3  = 220
AS3 = 233
B3  = 247
C4  = 262
CS4 = 277
D4  = 294
DS4 = 311
E4  = 330
F4  = 349
FS4 = 370
G4  = 392
GS4 = 415
A4  = 440
AS4 = 466
B4  = 494
C5  = 523
CS5 = 554
D5  = 587
DS5 = 622
E5  = 659
F5  = 698
FS5 = 740
```

```
G5  = 784
GS5 = 831
A5_ = 880
AS5 = 932
B5  = 988
C6  = 1047
CS6 = 1109
D6  = 1175
DS6 = 1245
E6  = 1319
F6  = 1397
FS6 = 1480
G6  = 1568
GS6 = 1661
A6  = 1760
AS6 = 1865
B6  = 1976
C7  = 2093
CS7 = 2217
D7  = 2349
DS7 = 2489
E7  = 2637
F7  = 2794
FS7 = 2960
G7  = 3136
GS7 = 3322
A7  = 3520
AS7 = 3729
B7  = 3951
C8  = 4186
CS8 = 4435
D8  = 4699
DS8 = 4978

# Bach Prelude in C.
bach = [
    C4, E4, G4, C5, E5, G4, C5, E5, C4, E4, G4, C5, E5, G4, C5, E5,
    C4, D4, G4, D5, F5, G4, D5, F5, C4, D4, G4, D5, F5, G4, D5, F5,
    B3, D4, G4, D5, F5, G4, D5, F5, B3, D4, G4, D5, F5, G4, D5, F5,
    C4, E4, G4, C5, E5, G4, C5, E5, C4, E4, G4, C5, E5, G4, C5, E5,
    C4, E4, A4, E5, A5_, A4, E5, A4, C4, E4, A4, E5, A5_, A4, E5, A4,
    C4, D4, FS4, A4, D5, FS4, A4, D5, C4, D4, FS4, A4, D5, FS4, A4, D5,
    B3, D4, G4, D5, G5, G4, D5, G5, B3, D4, G4, D5, G5, G4, D5, G5,
    B3, C4, E4, G4, C5, E4, G4, C5, B3, C4, E4, G4, C5, E4, G4, C5,
    B3, C4, E4, G4, C5, E4, G4, C5, B3, C4, E4, G4, C5, E4, G4, C5,
    A3, C4, E4, G4, C5, E4, G4, C5, A3, C4, E4, G4, C5, E4, G4, C5,
    D3, A3, D4, FS4, C5, D4, FS4, C5, D3, A3, D4, FS4, C5, D4, FS4, C5,
    G3, B3, D4, G4, B4, D4, G4, B4, G3, B3, D4, G4, B4, D4, G4, B4
]
```