Lab 5 GPIO, Interrupts, Concurrency (Part 1)

Name	SID	Checkoff

Objectives: Design an IoT app that combines analog and digital GPIO, interrupts and timers. Learn how to use an oscilloscope.

The application combines the following elements:

1. Buzzer playing tune
2. LED with PWM (pulse width modulation) dimming
3. Two-axis joystick and pushbutton serving the following functions (Part 2: next week's lab):
    - 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 the parts. This lab demonstrates many capabilities and may serve as a starting point for more sophisticated applications. After completing the lab, you are encouraged to improve the setup (ex: adjust the range of the joystick control for better sensitivity) and try other ideas. Or hook up a speaker for higher quality sound.

Parts and Tools

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

1 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.

A. 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 a resistor in series (to limit the current and avoiding frying the ESP32). A typical red led drops approx. 2V and draws about 15mA . Calculate the series resistor needed to achieve this with the 3.3V supply on the microcontroller board. 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.

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

2. 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.

3. 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!

B. Oscilloscope

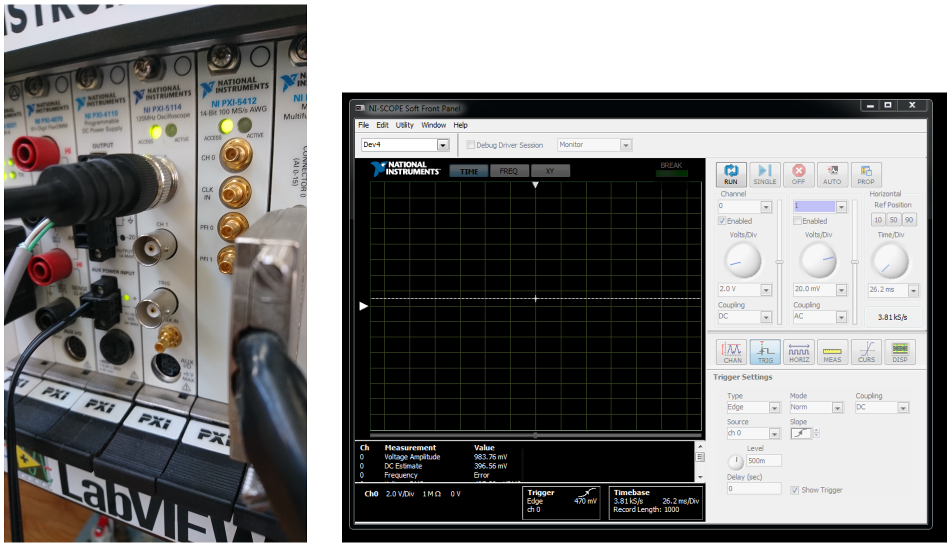
Write a program that configures two **different** PWM timers for 5 kHz and 8 kHz respectively, with 20 % and 60 % duty cycle. Have it ready in the lab so you can measure the signals with an Oscilloscope.

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.

In the 122 Hesse Lab we will use the available NI PXI-5114 Oscilloscope which provides a digital oscilloscope interface on the lab computer desktops. Grab a coaxial probe to connect to the PXI-5114 (fig. 1 (left)) and open the oscilloscope front panel by clicking on the indicated icon (fig. 2). The front panel provides many of the same functions as a benchtop oscilloscope (fig. 1 (right)). As with any new tool (oscilloscopes especially), some exploration with the interface is required to find the functions you need.



Dev4 · Debug Driver Session · Monitor

Channel: 0, Enabled, Volts/Div 2.0 V, Coupling DC
Channel: 1, Enabled, Volts/Div 20.0 mV, Coupling AC
Horizontal: Ref Position 10 50 90, Time/Div 26.2 ms, 3.81 kS/s

Trigger Settings: Type Edge, Mode Norm, Coupling DC, Source ch 0, Slope, Level 500m, Delay (sec) 0, Show Trigger

Ch	Measurement	Value
0	Voltage Amplitude	983.76 mV
0	DC Estimate	396.56 mV
0	Frequency	Error

Ch0 2.0 V/Div 1 MΩ 0 V
Trigger Edge ch 0 470 mV
Timebase 3.81 kS/s 26.2 ms/Div Record Length: 1000

Figure 1: Left: NI PXI-5114 with coaxial probe. Right: PXI Oscilloscope Front Panel Interface

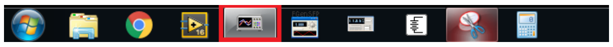


Figure 2: Oscilloscope on Windows Taskbar

Prelab Checkoff

Task	Checkoff
LED brightness demo and program	
Oscilloscope code	

2 Lab

A. 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: ☐

B. Playing a Tune

We configure a PWM channel to output the frequencies corresponding to a tune we want to play. A sample tune is offered 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. 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.

1. 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).
2. Use a for-loop to play a tune. Don't forget a sleep statement in the loop, or the tune will be very short!
3. 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!

Checkoff: ☐

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
]
```