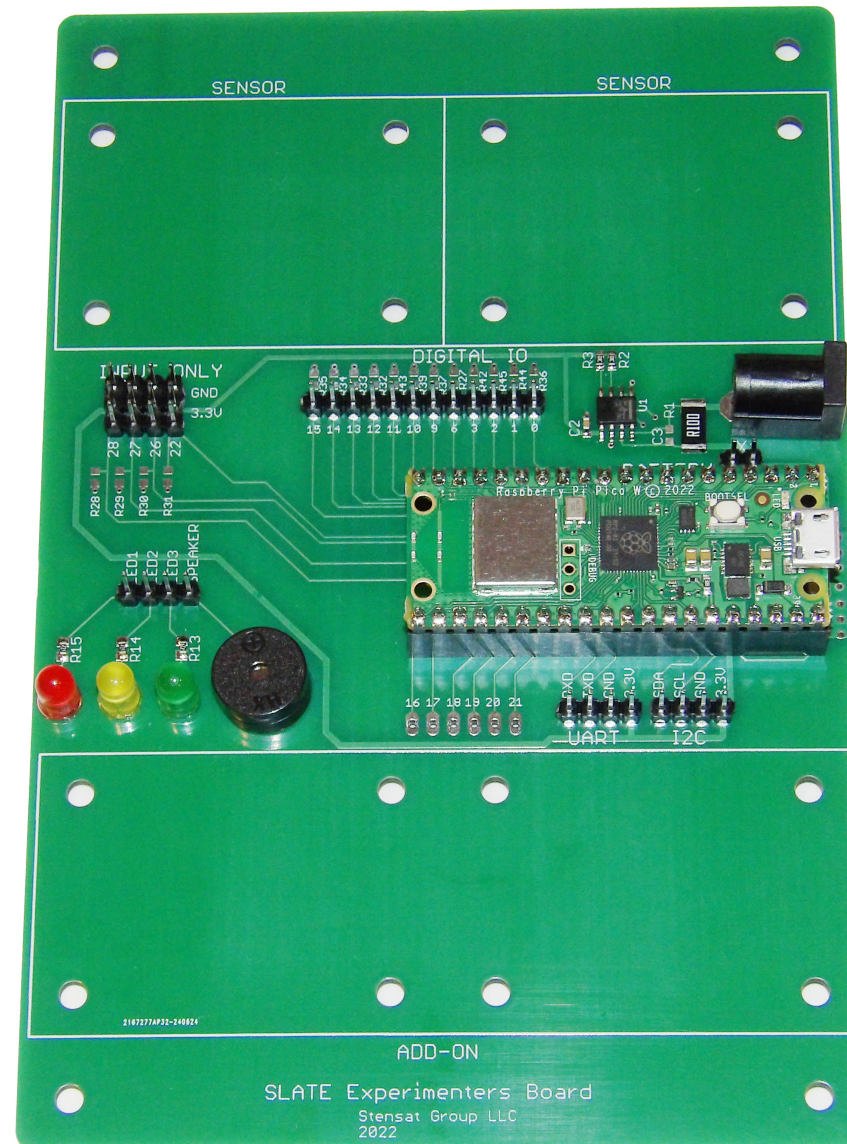


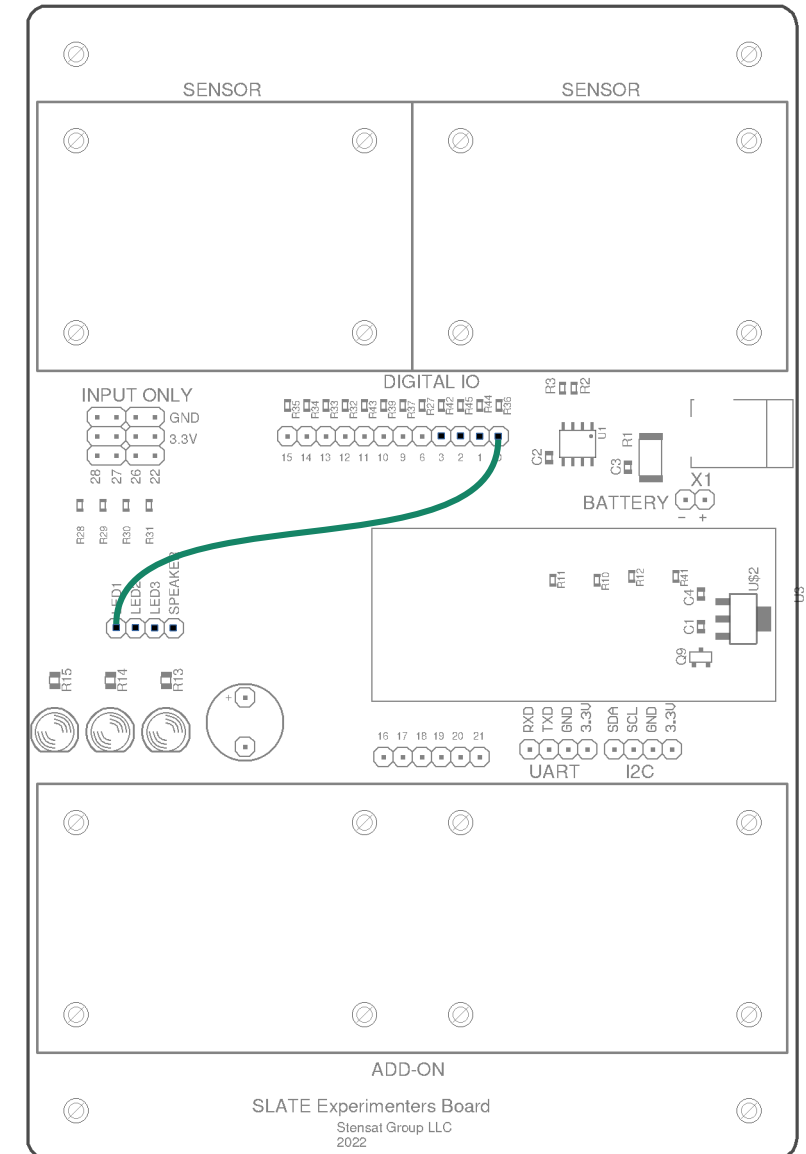
Digital Control



Controlling an LED

- Take a jumper wire and insert one end into pin GP0. Insert the other end into pin **LED1** as shown in the picture.
- Enter the following into the command shell:

```
from machine import Pin
gp0 = Pin(0,Pin.OUT)
gp0.on()
gp0.off()
```
- The LED should turn on when you enter the **gp0.on()** function and off when entering **gp0.off()**.



- The first line loads the Pin functions from the machine module. This adds commands to control the digital pins.
- The next line configures pin **GP0** to be an output. The function **Pin()** is used to configure digital pin 16 to be an output. It is assigned to object **d0**. **d0** can be any name.
- To set the digital pin 0 signal high, the function is called **on()**. To set it low, the function is called **off()**.
- **Experiment time:**
 - Pick another digital pin and connect it to the other LED. Pick a name for it and configure it so you can turn the LED on and off.
 - Enter the commands in the shell.

```
from machine import Pin

d0 = Pin(0, Pin.OUT)
d0.on()
d0.off()
```

- Enter the code to the right in the shell.
At the end of the last line, press the Backspace key and then press the Enter key.
- The LED should turn on for 1 second and turn off for 1 second.
- When you are done staring at the blinking LED, click on the **STOP** button to stop the program.
- A new module was imported to python. This one provides the function **sleep()** which stops the program executing for the number of seconds specified. You can also specify delays of less than a second. **0.1** will stop the program for a tenth of a second.

```
from machine import Pin
import time

d0 = Pin(0, Pin.OUT)
while 1:
    d0.on()
    time.sleep(1)
    d0.off()
    time.sleep(1)
```

- In the editor, click on **File** menu and select **New** and enter the program to the right.
- Save it using the name **blink.py**.
- Click on the **RUN** button to start the program. The LED will start blinking.
- Click on the **STOP** button to stop the program.
- Experiment time:
 - Change the sleep time so the LED blinks faster.
 - How fast can you get the LED to blink before you cannot see if blink.
- Experiment time two:
 - Add a second LED and have it blink opposite of the first LED. When the first LED is off, the second LED is on. When the first LED is on, the second LED is off.

```
from machine import Pin
import time

d0 = Pin(0, Pin.OUT)

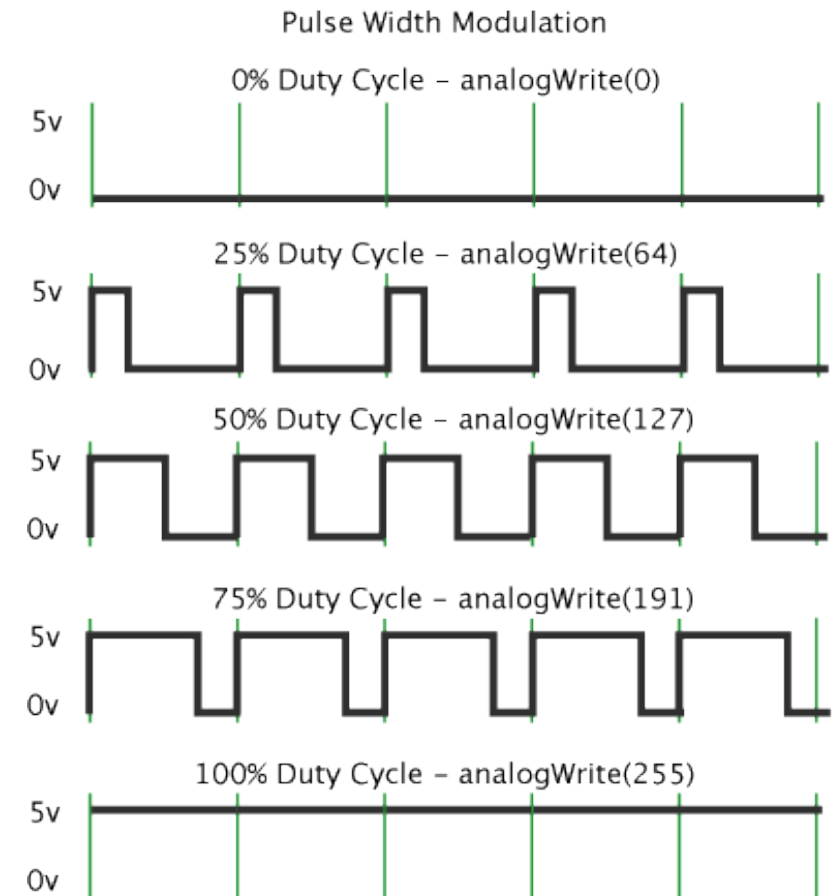
while 1:
    d0.on()
    time.sleep(0.5)
    d0.off()
    time.sleep(0.5)
```

Uploading Code and Managing Files

- Every time you change the code in the text editor, you need to save it. Generally the keys **Ctrl** and **S** will save the file using the same file name.
- Every time you want to run the program, click on the RUN button. Always click on the STOP button to stop the program.
- If you have a connection problem, try clicking on the STOP button and wait a few seconds. If that doesn't work, quit the Thonny program and restart it again.

Pulse Width Modulation

- Pulse width modulation is a way to control speed or intensity. Light dimmers use pulse width modulation to control the intensity of light.
- The speed of motors can be controlled the same way.
- The diagram to the right shows how pulse width modulation works. Basically, the longer the signal is on, the faster the motor or the brighter the light. The less the signal is on, the slower the motor or the dimmer the light.



- In the python shell with the LED still connected to **D0**, enter the commands below:

```
from machine import PWM,Pin
```

```
d0 = PWM(Pin(0),500,duty_u16=0)
```

- You need to import the PWM module to access the functions.
- The **PWM()** function configures the Pin **D0** to be a PWM port.
- The second parameter is how fast the pulses are generated.
- The third parameter sets the initial duty cycle.
- At this point, the LED is on. Now change the duty cycle.

```
d0.duty_u16(1023)
```

- This turns the LED on all the way. Now try:

```
d0.duty_u16(200)
```

- You will see the LED dim. Try other values. Realize the LED does not change linearly.

- In the editor, open a new program and enter the code to the right.
- This program introduces the **for** loop.
- Pin 0 is configured as a PWM port.
- The while loop will repeat forever.
- The for loop requires a variable which is **p**.
- The range is the range of values the for loop will assign to the variable **p** in sequence from 0 to 65535.
- The duty cycle is set with the value of **p**.
- **time.sleep()** stops the program for .001 seconds or 1 milliseconds.
- Upload the program and run it. The Led should cycle from dark to bright.

```
from machine import PWM, Pin
import time

d0 = PWM(Pin(0), 500, duty_u16=0)

while 1:
    for p in range(0, 65536):
        d0.duty_u16(p)
        time.sleep(.001)
```

- In the existing program, add a second LED.
- Have the LED increase in intensity like the first one.
- Once you have that working, figure out how to make the second LED go from bright to dark while the first one goes from dark to bright.
- Hints:
 - You do not add another for loop.
 - You need to do a little simple math.
 - Remember going from dark to bright, the value of **p** goes from 0 to 65535.
 - You want the second LED to go from 65535 to 0 using the variable **p** and a little math.
 - You can have the math equation in the argument of a function.

```
from machine import PWM, Pin
import time

d0 = PWM(Pin(0), 500)

while 1:
    for p in range(0, 65536):
        d0.duty_u16(p)
        time.sleep(.002)
```

- Write a program that turns one LED on and the other off and then does the opposite.
- Create a function that turns on or off both LEDs with two arguments.
 - Hint: It should look like this: `led(a,b)`
- Have the LEDs toggle one and off opposite of each other at a half second interval.

- The same PWM functions can be used to generate tones.
- Connect a jumper from Pin 2 to the speaker pin.
- Enter the program to the right and call it **tone1.py**.
- Pin 2 is set up as an PWM output.
- The duty cycle is set to 50% with the value of 32767. 32767 is half the maximum value allowed for the duty cycle. so a square wave is generated.
- Next, the frequency is set to 500 Hz.
- The tone is generated for 2 seconds and turned off by setting the duty cycle to 0. This disables the toggling of the pin. The frequency cannot be set to zero Hz.

```
from machine import PWM, Pin
from time import sleep

tone = PWM(Pin(2))
tone.duty_u16(32767)

tone.freq(500)
sleep(2)
tone.duty_u16(0)
```

- Try this program to annoy your friends. Save it as **tone_annoy.py**.
- The for loop sequences through the range of frequencies from 100 to 3000 Hz. The sleep function adds a 100 microseconds delay between updates.
- The while loop repeats the for loop forever.
- Experiment with different speeds and tone ranges. Try making the tone go from high to low.

```
from machine import PWM, Pin
from time import sleep

tone = PWM(Pin(2))
tone.duty_u16(32767)

while 1:
    for i in range(100, 3000):
        tone.freq(i)
        sleep(0.0001)
```

- A tune can be played by using a list of tones and duration of the tones.
- This example uses two data structures, tuples and a list.
- The tuple will contain the tone frequency and the duration of the tone.
- The list will contain the tuples.
- In this example, the tuple is indicated by using ().
- The first element of the tuple is the frequency and the second element is the time duration.
- The list contains a series of tuples.
- The for loop sequences through the list.
- Each element of the tuple can be accessed similar to how list elements are accessed.

```
from machine import Pin,PWM
import time

t = PWM(Pin(2))
t.duty_u16(32767)

tones = [(500,0.5),(300,0.25),(600,.5),(800,1)]

for i in tones:
    t.freq(i[0])
    time.sleep(i[1])
t.duty_u16(0)
```

