Solución Actividad 1

Una vez instalado el SDK de Moddable trata de realizar los siguientes programas que manejan las entradas/salidas digitales y analógicas.

1. **Hacer un programa que haga que 2 leds (en los pines 2 y 16, por ejemplo) parpadeen alternativamente.**

Como vamos a utilizar entrada y salida digital debemos tener un fichero `manifest.json` con el siguiente contenido:

```
1 {
2   "include": [
3     "$(MODDABLE)/examples/manifest_base.json",
4     "$(MODULES)/pins/digital/manifest.json"
5   ],
6   "modules": {
7     "*": "./main"
8   }
9 }
```

El código del fichero `main.js` podría ser el siguiente:

```
 1 import Timer from "timer";
 2 import Digital from "pins/digital";
 3
 4 debugger;
 5
 6 let led2 = new Digital({
 7   pin: 2,
 8   mode: Digital.Output
 9 });
10
11 let led16 = new Digital({
12   pin: 16,
13   mode: Digital.Output
14 });
15
16 let count = 0;
17 Timer.repeat(() => {
18   trace(`En cuenta ${++count} \n`);
19   led2.write(count & 1);
20   led16.write((~count) & 1);
21 }, 1000);
```

Se comienza se declaran los objetos que gestionarán los pines 2 y 16 como salidas digitales (líneas 6 a 14). Usando `Timer` definimos una función anónima, que se ejecuta cada 1000 milisegundos (líneas 17 a 21). Ésta cambia el valor de salida de cada led en función del valor del

bit menos significativos de contador `count` (líneas 19 y 20). Además se envía por la serial mensaje que indica el valor de dicha variable (línea 18).

2. **Detectar cuando un botón (conectado al pin 0, por ejemplo) es pulsado y mandar mensaje por la serial.**

El fichero `manifest.json` para este caso es el mimo que eh ejercicio anterior.

En cuanto al fichero `main.js` se pueden abordar dos soluciones. La primera podría utilizar una escaneo periódico del pin y determinar cuando su valor (digital) es distinto del anterior y así determinar que se ha producido un cambio:

```
 1 import Timer from "timer";
 2 import Digital from "pins/digital";
 3
 4 const BUTTON_PIN = 0;
 5
 6 const bot0 = new Digital({
 7   pin: BUTTON_PIN,
 8   mode: Digital.InputPullUp,
 9 });
10
11 trace(`Botón en el pin ${BUTTON_PIN}\n`);
12
13 let anterior = 0;
14 Timer.repeat(() => {
15   const actual = bot0.read();
16   if (actual !== anterior) {
17     trace(`Cambió botón a ${actual}\n`);
18     anterior = actual;
19   }
20 }, 100);
```

Pero más eficiente es usar la clase `Monitor` para que se invoque un callback cuando se produzca un cambio en el pin:

```
 1 import Digital from "pins/digital";
 2 import Monitor from "pins/digital/monitor";
 3
 4 debugger;
 5
 6 const BUTTON_PIN = 0;
 7
 8 const bot0 = new Monitor({
 9   pin: BUTTON_PIN,
10   mode: Digital.InputPullUp,
11   edge: Monitor.Rising | Monitor.Falling,
12 });
13
```

```
14
15 let contador = 0;
16
17 bot0.onChanged = function() {
18   trace(`Botón estado ${this.read()} → ${++contador}`
19     + ` → ^ ${this.rises} v ${this.falls}\n`);
20 }
21
22 trace(`Botón en el pin ${BUTTON_PIN}\n`);
```

Además el objeto de la clase `Monitor` lleva la cuenta del número de flancos que ha sufrido el pin. Sus valores se muestran en línea 19.

Tener en cuenta que es necesario incluir `monitor` en el `manifest.json`:

```
 1 {
 2   "include": [
 3     "$(MODDABLE)/examples/manifest_base.json",
 4     "$(MODULES)/pins/digital/manifest.json",
 5     "$(MODULES)/pins/digital/monitor/manifest.json"
 6   ],
 7   "modules": {
 8     "*": "./main"
 9   }
10 }
```

3. **Leer un valor analógico (en el pin analógico 0, por ejemplo) y mandar el valor de su medida por la serial**

Tenemos que hacer uso de la clase `Analog`. Por ello hay que incluirla en el `manifest.json`:

```
1 {
2   "include": [
3     "$(MODDABLE)/examples/manifest_base.json",
4     "$(MODDABLE)/modules/pins/analog/manifest.json"
5   ],
6   "modules": {
7     "*": "./main"
8   }
9 }
```

En nuestro `main.js` bastará lanzar un temporizador que repetidamente lea el valor del pin lo envíe por la serial con la función `trace`:

```
1 import Analog from "pins/analog";
2 import Timer from "timer";
3
4 debugger;
5
```

```
// no es pin GPIO sino correspondiente a ADC-1
const PIN_ADC = 0;

Timer.repeat(() => {
  let valor = Analog.read(PIN_ADC);
  trace(`Valor leído ${valor}\n`);
}, 500);
```

4. **Dado un led rojo en el pin 2 y uno verde en el pin 16 hacer que vayan disminuyendo y aumentando su intensidad de manera sucesiva. Comiencen ambos encendidos, se vaya apagando el rojo, cuando esté apagado el rojo comience a apagarse el verde, cuando esté apagado el verde comience a encenderse el verde, y así sucesivamente.**

Para variar la intensidad de los led tenemos podríamos utilizar la salida digital y hacer, usando temporizadores, que los leds parpadeen con distintos periodos a 0 y 1. Eso precisamente es lo que nos proporciona la clase PWM que aprovecha los PWMs que tiene el ESP32 en hardware.

Tendemos que incluir la clase PWM en el `manifest.json`:

```
{
  "include": [
    "$(MODDABLE)/examples/manifest_base.json",
    "$(MODDABLE)/modules/pins/pwm/manifest.json"
  ],
  "modules": {
    "*":  "./main"
  }
}
```

A la vista del enunciado, nuestro programa tiene que pasar por cuatro estados según que led esté encendiéndose y apagándose. Para ello implementamos una pequeña máquina de estados: en la variable `estado` tendremos el estado actual y mediante un `switch` repartimos las instrucciones de cada estado.

Una posible solución sería:

```
import Timer from "timer";
import PWM from "pins/pwm";

debugger;

const r = new PWM({ pin: 2 });
const g = new PWM({ pin: 16 });

let rVal = 1023;
let gVal = 1023;

r.write(rVal);
```

```
13 g.write(gVal);
14
15 let estado = 0;
16
17 Timer.repeat(() => {
18   switch (estado) {
19     case 0:
20       rVal -= 20;
21       if (rVal < 1) {
22         rVal = 1;
23         estado = 1;
24         trace(`Pasamos a estado ${estado}\n`);
25       }
26       r.write(rVal);
27       break;
28     case 1:
29       gVal -= 20;
30       if (gVal < 1) {
31         gVal = 1;
32         estado = 2;
33         trace(`Pasamos a estado ${estado}\n`);
34       }
35       g.write(gVal);
36       break;
37     case 2:
38       rVal += 20;
39       if (rVal > 1023) {
40         rVal = 1023;
41         estado = 3;
42         trace(`Pasamos a estado ${estado}\n`);
43       }
44       r.write(rVal);
45       break;
46     case 3:
47       gVal += 20;
48       if (gVal > 1023) {
49         gVal = 1023;
50         estado = 0;
51         trace(`Pasamos a estado ${estado}\n`);
52       }
53       g.write(gVal);
54       break;
55   }
56 }, 50);
```