Hands on session 08: Timers & PWM

Session Introduction: This session will introduce the concept of Timers & PWM

Session Objectives:

- **To understand how timers work**
- **To understand how PWM works**

Timers

A Timer is an internal interrupt that can trigger an alarm and an associated action at a specific time repeatedly. A Timer is considered an interrupt because it "interrupts" the main thread to execute the code associated with it. Once the associated code has been executed, the program continues where it left off.

ESP-32 Timers

The ESP32 contains two groups of hardware timers. Each group has two general purpose hardware timers, so the ESP32 has a total of 4 timers which are numbered 0-3. These are all generic 64-bit based timers. Each Timer has a 16-bit Prescaler (from 2 to 65536) as well as 64-bit ascending / descending counters which can be automatically reloaded.

Prescaler

The Timer uses the processor clock to calculate the elapsed time. It is different for each microcontroller. The quartz frequency of the ESP32 is 80MHz.
The ESP32 has two groups of timers. All timers are based on 64-bit Tic counters and 16-bit time dividers (prescaler). The prescaler is used to divide the frequency of the base signal (80 MHz for an ESP32), which is then used to increment or decrement the timer counter.
To count each Tic, all you have to do is set the prescaler to the quartz frequency. Here 80.

The timer simply counts the number of Tic generated by the quartz. With a quartz clocked at 80MHz, we will have 80,000,000 Tics.
By dividing the frequency of the quartz by the prescaler, we obtain the number of Tics per second
80,000,000 / 80 = 1,000,000 tics / sec

Configure Timers

In order to configure the timer, we will need a pointer to a variable of type **hw_timer_t .**

hw_timer_t * timer = NULL;

The **timerBegin(id, prescaler, flag)** function is used to initialize the Timer. It requires three arguments

- id the Timer number from 0 to 3
- prescale the value of the time divider
- flag true to count up

timer = timerBegin(0, 80, true);

Handling function
Handling function will be executed when the interrupt is generated. This is done with a call to the **timerAttachInterrupt()** function.

pass our global timer variable as first input, as second the pass ISR, and as third the value **true**, so the interrupt generated is of edge type.

Trigger an alarm

Once the Timer is started, all that remains is to program an alarm which will be triggered at regular intervals. For this we have the **timerAlarmWrite(timer, frequency, autoreload)** method which requires 3 parameters.

- timer the pointer to the Timer created previously
- frequency the frequency of triggering of the alarm in ticks.
- autoreload true to reset the alarm **automatically** after each trigger.

timerAlarmWrite(timer, 1000000, true);

Finally, we start the alarm using the **timerAlarmEnable(timer)** method.

timerAlarmEnable(timer);

PWM

The ESP32 has a LED PWM controller with 16 independent channels that can be configured to generate PWM signals with different properties.

Here's the steps you'll have to follow to dim an LED with PWM using the Arduino IDE:

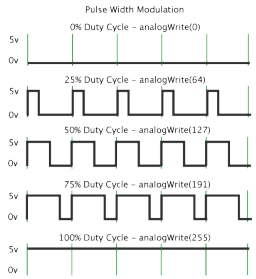
1. First, you need to choose a PWM channel. There are 16 channels from 0 to 15.
2. Then, you need to set the PWM signal frequency.
3. You also need to set the signal's duty cycle resolution. We'll use 8-bit resolution, which means you can control the LED brightness using a value from 0 to 255.

ledcSetup(ledChannel, freq, resolution);

4. Next, you need to specify to which GPIO or GPIOs the signal will appear upon. For that you'll use the following function:

```
ledcAttachPin(GPIO, channel)
```

This function accepts two arguments. The first is the GPIO that will output the signal, and the second is the channel that will generate the signal.



5. Finally, to control the LED brightness using PWM, you use the following function:

```
ledcWrite(channel, dutycycle)
```

This function accepts as arguments the channel that is generating the PWM signal, and the duty cycle.