# Embedded System Lab 2

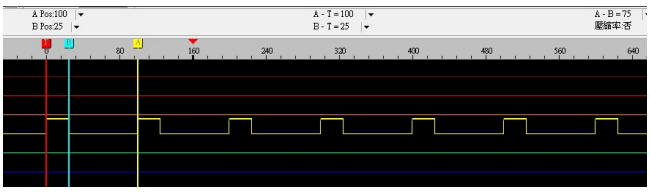
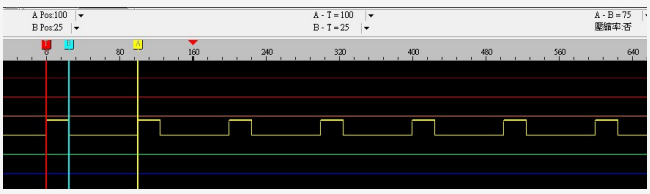
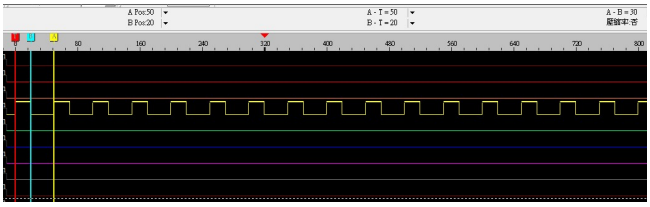
## PWM Applications

B07901095 吳宜庭

## I. Basic functions of Mbed PWM

- `period`: specifies the PWM period, the unit is second
- `write`: duty cycle of high signal within the period
- =: direct assign the pwm node to the value of duty cycle (same function as write)
- `pulsewidth`: specifies the pulse-width in second of high signal

Resources: https://os.mbed.com/handbook/PwmOut

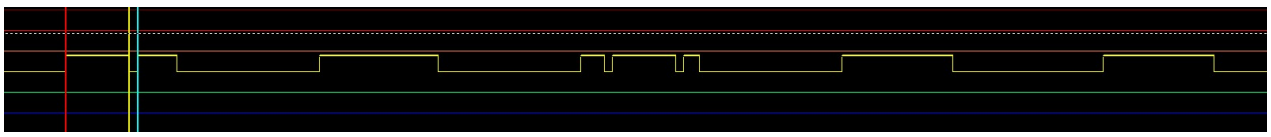
| | | |
|---|---|---|
| wave (a) | `speaker.period(0.1f);`<br>`speaker = 0.25;`<br>`wait_us(10);` |  |
| | `speaker.period(0.1f);`<br>`speaker.write(0.25f);`<br>`wait_us(10);` |  |
| wave (b) | `speaker.period(0.05f);`<br>`speaker.pulsewidth(0.02f);`<br>`wait_us(10);` |  |

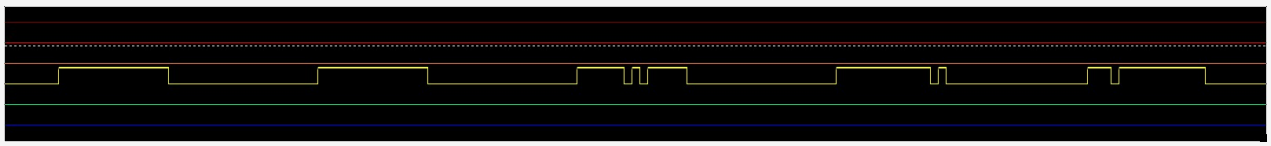
## II. Application

There are many applications related to speakers on mbed and PWM output. Once connected to a speaker, we can use `pwm_period` to determine the frequency of sound——a smaller period results in a higher frequency and thus a higher pitch. The duty cycle of PWM affects how loud the volume is, with 100% duty cycle being the loudest. We also use `wait_us()` to hold the pwm period for the time of the beat.

The notes and beats are written as an array at the beginning of the code. Mbed will run through the whole set of notes and beats, and generate a waveform containing all the different periods and wait times.

A snippet of the waveform of “Happy Birthday Song” is as follows:





Resources: https://os.mbed.com/users/ffxx68/code/Speaker_test/docs/7fdec2560f78/main_8cpp_source.html

## III. Review Questions

1. Why does MCU SOC need a clock tree?

There are a lot of processes taking place on the MCU simultaneously, each requiring a different clock cycle. Thus, it is difficult to control everything with the same System Clock. The clock tree helps to separate the System Clock into smaller frequencies clocks, enabling more dynamic resource allocation and energy preservation.

2. Why so many clock sources are required or available in a MCU SOC?

Having many clock sources allows different processes to choose different clock cycles based on the properties. For example, idling processes or processes at sleep can choose a lower frequency clock to be more energy-efficient.

## IV. Discussions

The biggest problem I ran into is not knowing how to precisely modify the pwm period, utilizing both `pwm.period()` and `wait_us()`. I tried the simple code changing periods, but only the effect of the later part was observed.

For instance, in the leftmost figure below, the output waveform is the same as wave (a) in section I, in which only the 0.1sec period can be observed. Similarly, in the middle figure, the output waveform is the same as wave (b) in section I, in which only the 0.05 sec period can be observed.

One of my guesses was that the pwm output and the LogicCube do not match each other in frequencies. Thus, I experimented with a for loop as the rightmost figure shows. However, the output waveform was still the same as wave (b) in section I. Considering the application in section II that runs successfully, the problem might be that the waiting time is too short.

| | | |
|---|---|---|
| ```<br>speaker.period(0.05f);<br>speaker.pulsewidth(0.02f);<br>wait_us(10);<br><br>speaker.period(0.1f);<br>speaker.write(0.25f);<br>wait_us(10);<br>``` | ```<br>speaker.period(0.1f);<br>speaker = 0.25;<br>wait_us(10);<br><br>speaker.period(0.05f);<br>speaker.pulsewidth(0.02f);<br>wait_us(10);<br>``` | ```<br>for(int i = 0; i < 10000; ++i){<br>    speaker.period(0.1f);<br>    speaker = 0.25;<br>    wait_us(10);<br>}<br><br>speaker.period(0.05f);<br>speaker.pulsewidth(0.02f);<br>wait_us(10);<br>``` |
| Only see 0.1 sec period. | Only see 0.05 sec period. | Only see 0.05 sec period. |
| Wave (a) | Wave (b) | Wave (b) |