Lab Sheet 6: Timer Modules

See deadlines on QMPlus

1 Aims

The aim of this lab is to introduce the PIT (Programmable Interrupt Timer) and the TPM (Timer/PWM Module).

- Understand how to use the PIT for precise timing.
- Understand how to use the TPM to implement PWM as a digital form of analogue output.

1.1 Overview of Activities

Perform the following activities and answer the questions given in section 5.

1. Activity 1: Download and run the sample project, connecting the hardware.
2. Activity 2: Tone steps: using the PIT.
3. Activity 3: Volume steps: using PWM.

You must also answer the questions (see Section 5) on the QMPlus quiz and demonstrate your program in the scheduled lab time. *You are recommended to look at the questions as you go along: some are about the given code and answering them will help you complete the rest of the lab.*

1.2 API Notes

The following notes are included in this sheet:

1. Appendix A: Use of the PIT to Generate an Audio Tone
2. Appendix B: Use of the PWM to Control the Volume

2 Activity 1: Download, Read, Connect and Run the Sample Project

Download the sample project from the QMPlus page. The following are provided:

- An API[1] for using the PIT.
- An API for using the PWM function of the TPM.

The example project shows the use of the two APIs:

1. Generates a fixed tone at half volume.
2. A button, connected between PTD6 (J2 pin 17) and ground (J2 pin 14), switches the tone on and off.
3. The RTOS is used:
    a. Task `t_tone` (function `toneTask`) controls the state of the tone.
    b. Task `t_button` (function `buttonTask`) signals the toneTask when the button is pressed.

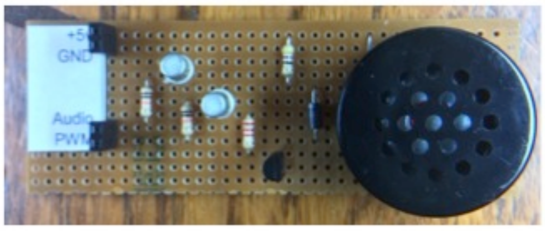
The example program uses a circuit that drives a speaker, with inputs for an audio signal (using PIT) and a volume control (using PWM).

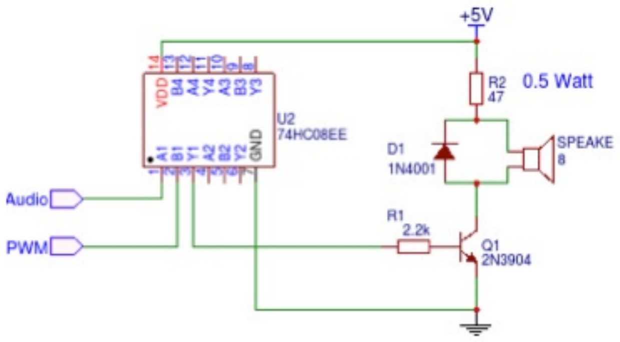
Pause and read the complete code carefully before going on. *Answer the questions on the question sheet about the code.*

[1] Note that the two APIs can be used in projects without the RTOS if you prefer.

2.1 Hardware Connections

The audio hardware and the circuit implemented[2], are shown below.





The connections are:

Name	KL25Z Pin Label	Purpose
PWM	PTA4	Volume control using PWM
Audio	PTA2	Audio signal
+ve	P5V_USB	5v positive supply
GND	GND	0v

3 Activity 2: Tone Steps

The aim of this activity is to change the system so that the button changes the tone, rather than switching the tone on and off. The steps needed are:

1. Determine the periods (i.e. cycle counts) of a set of tones (for example MIDI tones 60 through to 95, which is three octaves), using the formula in Appendix A.
2. Store the periods in a `const uint32_t` array.
3. On every button press, use the next period from the array, returning eventually to the start.

Viva Required [8 marks]

A demonstrator will witness the viva. Ensure that both you and the demonstrator sign the viva sheet.

Note: you may make a single demonstration of your work at the end of activity 3 if you prefer.

The code should be uploaded – see instructions on QMPlus.

[2] The circuit diagram is included for those interested; there is no requirement to understand it. Unfortunately, the built circuit does not correspond exactly to the design shown in the diagram. The build version is simpler and adequate.

4 Activity 3: Volume Steps

The aim of this activity is to add an additional button to the system so that the volume as well as the tone (see activity 2) can be adjusted. The steps needed are:

1. Choose a port/pin for the button and connect it.
2. Create a task that is used to monitor the button. Ideally, enhance the existing buttonTask function so that it can handle more that just one button. Think about how to do this without duplicating lots of code.
3. The volume is controlled by the function `setPWMDuty()` which has a duty parameter in the angle 0-128. The perception of volume is non-linear so you may wish to try doubling the duty cycle (0, 1, 2, 4, 8, 16 …), going back to zero when the maximum is reached.
4. Create a new task to control the volume. Signal it when the new button is pressed and adjust the volume.

Viva Required [8 marks]

A demonstrator will witness the viva. Ensure that both you and the demonstrator sign the viva sheet.

The code should be uploaded – see instructions on QMPlus.

5 Question Sheet

Answers to these questions should be completed on the QMplus quiz.

5.1 Questions About the Given Code [2 marks each]

Relates to	**Questions**
Section 2 Activity 1	Give the line of code in main.c (in the provided project) that determines the frequency of tone. Explain briefly how it works
Section 2 Activity 1	Give the name of the function (not containing any other function calls) that is called to generate the audio tone in the provided project. Explain how it is called.
Section 2 Activity 1	Give the line of code (in main.c) that is called to determine the volume in the provided project. What change would you make to increase the volume?

5.2 Question about concepts

Answer the following questions concisely:

1. Explain the principle of PWM for controlling the volume of the audio tone.
2. The PWM modulation frequency is approximately 47 KHz. a) Explain why this frequency is acceptable when PWM is being used to control the volume of an audio tone. b) The PWM is instead to be used to control the brightness of an LED: explain whether the modulation frequency could be lower or would need to be higher than 47 KHz.

[8 marks]

6 Appendix A: Using the PIT to Generate an Audio Tone

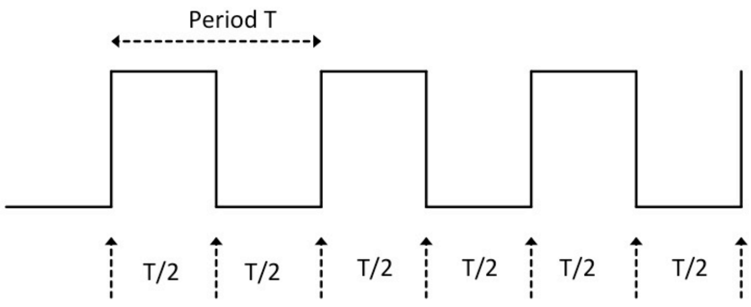
6.1 Principles

The KL25Z has two PIT channels. Each channel has a 32-bit counter. The counter is loaded with a value: it then counts down to zero, generates an interrupt and reloads the value.

The PIT can therefore be used to generate an interrupt at precise intervals. The value loaded to the counter controls the interval. The counter is clocked at 24 MHz, so the count is in units of 1/24 microsecond (41.7 nanoseconds), giving a maximum delay of 178.9 seconds. For longer delays the two counters can be combined to give a 64-bit counter, but this configuration is not covered here.

6.2 Example Calculation of Frequency

Recall that period (T) = 1 / frequency (f) and that we need to toggle the output every T/2.



Suppose you wish to create a sound at 440 Hz (which I read is concert pitch for the A above middle C). Then f = 440 Hz, we can determine the number of cycles of 24 MHz clock in the interval T/2 (where T is 1/f):

$$\frac{T}{2} = \frac{24{,}000{,}000}{440 \times 2} = \; 27{,}273 \; cycles$$

6.3 MIDI Tuning Standard

The MDI tuning standard (see https://en.wikipedia.org/wiki/MIDI_Tuning_Standard) uses this as a reference frequency and uses frequency intervals of the 12th root of 2, with 12 notes to an octave. 440 Hz is MIDI note 69; the formula for frequency of MIDI note ‘d’ is:

$$f = 2^{(d-69)/12} \times 440 Hz$$

Combining this with the formula above, we get N, the number of cycles of a 24 MHz clock in half the period of MIDI note d, is given by:

$$N = \frac{12{,}000{,}000}{2^{(d-69)/12} \times 440} \text{ cycles}$$

Use this formula to calculate the PIT counts for a range of notes. It is recommended that you do this offline (cf. in the program), for example using a spreadsheet and put the result in the program as constants.

6.4 Using the PIT Software

The PWM software is in two files:

1. PIT.c contains the code for the API functions and the ISR.
2. PIT.h is a header file.

The following functions are provided:

Function	Description
`void configurePIT(int c)`	Configure the PIT on channel c. Interrupts are enable but the timer is not started.
`void startTimer(int c)`	Start the timer on channel c.
`void stopTimer(int c)`	Stop the timer on channel c.
`void setTimer(int c, uint32_t t)`	Set the timer. A new count down value can be set without stopping the timer; in this case, it used when the current value next expires.

6.5 Interrupt Service Routine (ISR)

As well as the functions above, an ISR is needed. There is a single ISR for both channels: you need to look at and reset the flag that shows that a particular counter has run down to zero. The demonstration project includes the necessary code in file PIT.c: you need to remove the application specific part and replace it with you own code.

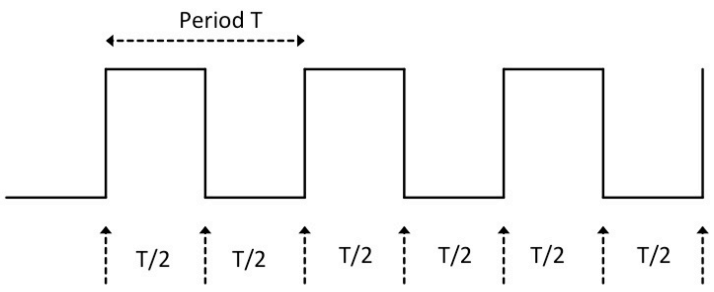
6.6 Pin Usage

There is no pin associated with the PIT. Here it is used to generate a periodic interrupt: the interrupt handler (ISR) toggles a GPIO pin.

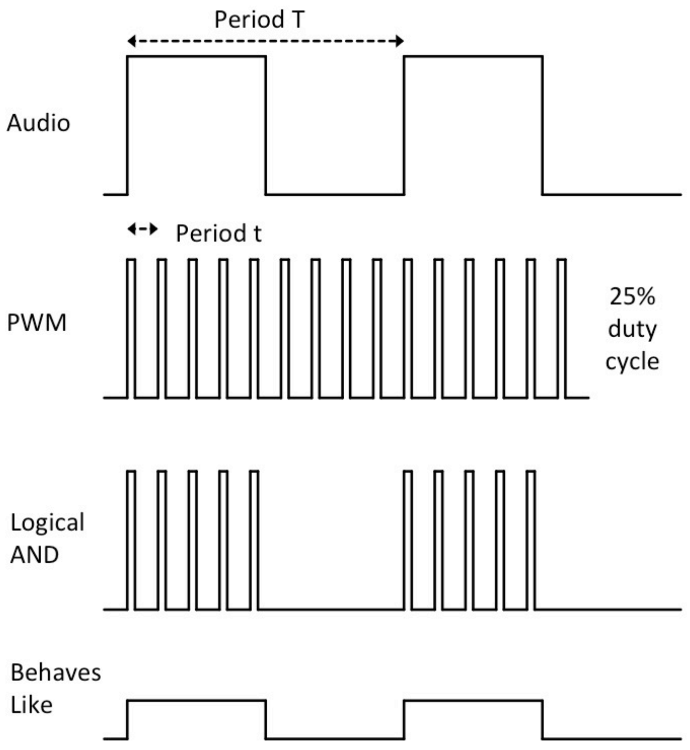
7 Appendix B: Volume Control using PWM

7.1 Principles

Suppose that a tone is being generated by switching an output from high to low at a fixed frequency.



The volume can be varied by using pulse width modulation (PWM). The principles of PWM are shown below. We require $f = 1/t$ to be much greater than the audio frequency F (which is 1/T).



The KL25Z includes a TPM module that can be configured for PWM.

7.2 Use of TPM Module for PWM

The PWM software is in two files:

1. TPM_PWM.c contains the code
2. TPMPWM.h is a header file

Two functions are provided:

1. `configureTPM0forPWM()` which configures the TPM 0 channel 1 for PWM output.
2. `setPWMDuty(unsigned int duty)`, which changes the volume. The duty parameter is from 0 (off) to 128 (max).