Lab Sheet 4: Timer Modules

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.

PWM (pulse width modulation) is used to vary the brightness of the LEDs on the development board. In Lab 1 we saw how different colours can be created by mixing the colours of the three colour LED. A greater range of colours can be created if we also vary the brightness.

1.1 Overview of Activities

Perform the following activities.

1. Activity 1: Download and run the sample project, connecting the hardware.
2. Activity 2: Modify the sequence of LED colours
3. Activity 3: Control the timing of the colour sequence
4. Activity 4 (Bonus): Two Patterns

Assessment. If you wish, you can present work on this lab exercise for assessment. Complete any of the following steps (see QM+ for full details and deadlines)

- **Answer the questions on the QMPlus quiz.** *You are recommended to look at the questions about the given code as you go along as they may help you complete the rest of the lab.*
- **Demonstration in the lab.** Ask one of the demonstrators to see your code working during the scheduled lab times.
- **Code assessment.** Complete the QM+ assignment to indicate that your code is to be assessed, provided you have demonstrated it. Do not attach any documents. The only way to submit code is to update your repository on GitHub. You can push code as often as you wish, whether or not you would like it assessed.

1.2 API Notes

The following notes are included in this sheet:

1. Appendix A: Use of the PIT to for timing
2. Appendix B: Use of the PWM to brightness

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

2.1 Download the Sample Project

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

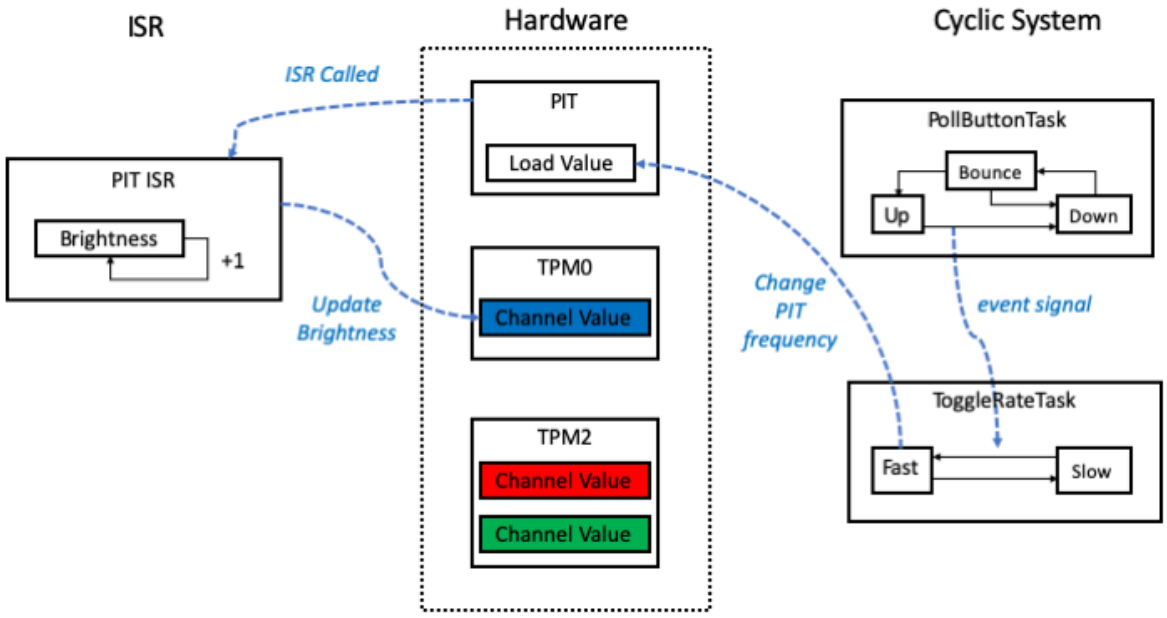
- An API for using the PIT.
- An API for using the PWM function of the TPM.

The example project shows the use of the two APIs:

1. The green LEDs is cycled through 32 different brightness levels
2. The changes in brightness are made in the PIT ISR. This is in the main.c file.
3. Button B1 is polled. When the button is pressed a signal is sent to the 'ToggleRate' task.

4. The task 'ToggleRate' switches between two rates: a fast one and a slow one. The timing is controlled by the PIT load value.

The structure of the program is quite complex. Find all the parts and understand how they relate. The diagram below depicts the structure.

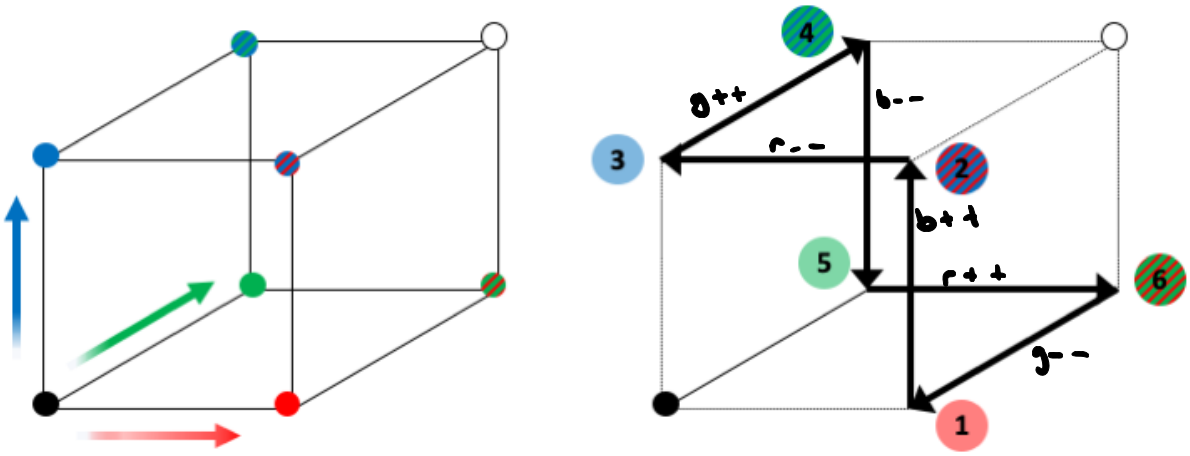


2.2 Make Trivial Modifications

To check you understand how the parts of the code relate, try making some changes. For example, change the LED colour or combine two LEDs.

3 Activity 2: Modify the sequence of LED Colours

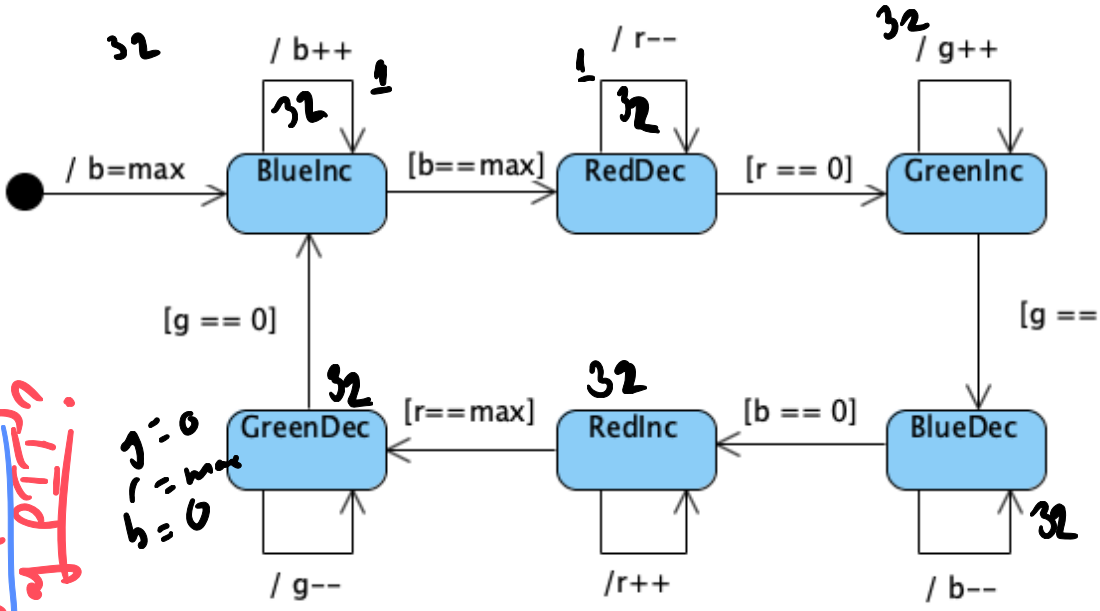
The aim of this activity is to change the system to display a sequence of colours. Each colour has 32 levels of brightness (including zero brightness, which is off). As there are three colours we can map the combinations into a cube, as shown on the left-hand side below.



The requirements are to move on the trajectory shown on the right-hand side above, and in the state transition model below.

- At the nodes of the cube 1, 3 and 3, just one colour of the LED is lit and has full brightness.
- At the nodes of the cube 2, 4 and 6, two colours of the LED are lit with full brightness.
- Moving between nodes, the brightness of one of the LED colours is increased or decreased in steps.
- Start with the red full on (position 1) and increase the blue brightness in steps

You should implement the following state transition diagram to meet these requirements. Create a separate function (which will be like the task functions) but call it from the ISR rather than the main loop.



Code changes: you should commit changes to the local repository. You can do this now and then commit further changes later. Also push committed changes to GitHub.

4 Activity 3: Control the Timing of the Colour Sequence

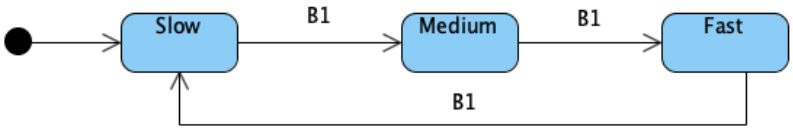
The aim of this activity is to control the time needed to complete the cycle of the cube. The required rates are given in terms of the time taken to go from position 1 and back, in the cube above.

Rate	Total Time	Step Time	Load Value
Slow	9 s		
Medium	5 s		
Fast	2 s		

Complete the table above. The steps needed are:

1. Given that there are 32 levels of brightness, calculate the total number of transitions required to complete the cycle.
2. Use this and the total times to calculate the step times.
3. Use the information in the code (and Appendix A) to calculate the PIT load value need to achieve the correct timings. Check using a stop watch.

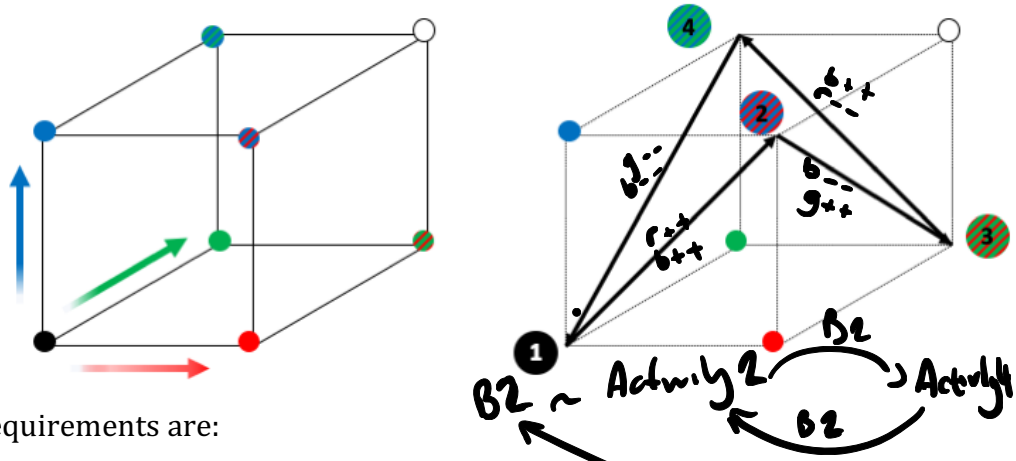
Button B1 is used to control the rate. The state transition model for the 'ToggleRateTsk' is shown below:



5 Activity 4 (Bonus): Two Patterns

With 32 brightness levels for the three LEDs there are a total of 32^3 combinations (i.e. around 32,000), so there are many more patterns we could make.

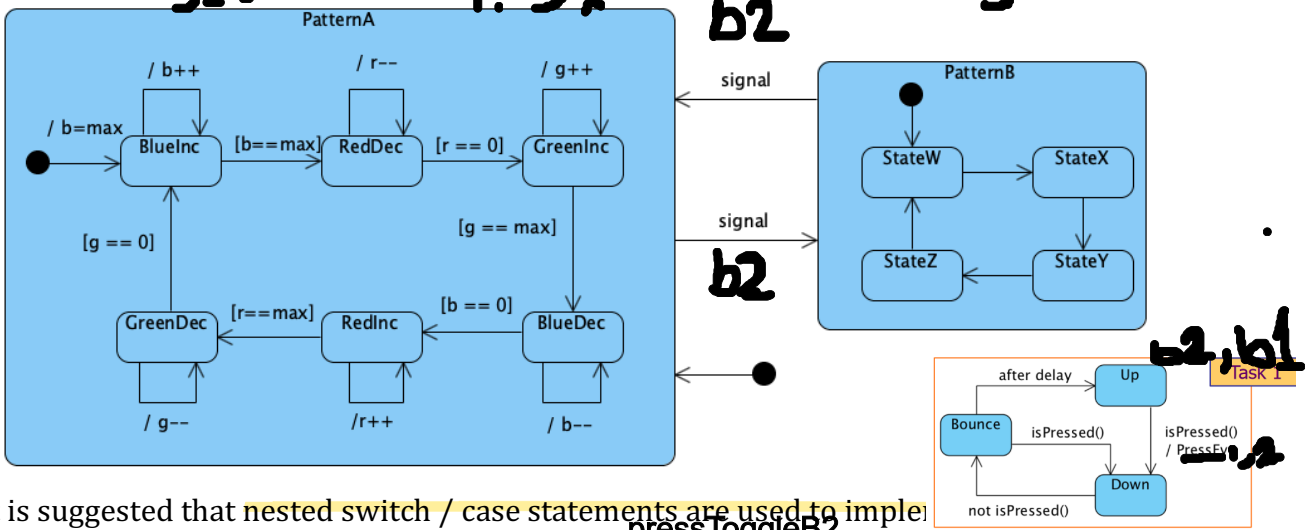
A second pattern is shown below, with 4 moves across the faces of the cube. These moves require the brightness of two LEDs to be updated together.



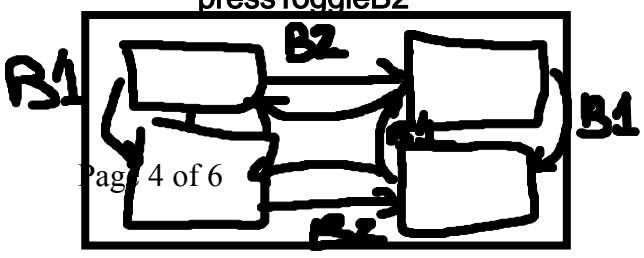
The requirements are:

- To move in the pattern shown above, from '1' (all colours off) to '2' (red and blue full on) and so on, as shown.
- Button B2 is pressed to change between patterns.
- The rate (i.e. total time) at which the pattern is completed remains unchanged until B1 is next pressed. [**Hint**: since the number of steps changes, the PIT load values are different even though the total time is unchanged.]

You should follow the *incomplete* design below, completing the information that is missing.



It is suggested that nested switch / case statements are used to implement the transition model.



6 Appendix A: Using the PIT to Control Timing

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 half the CPU clock speed. The CPU clock has frequency 20,971,520 Hz, so the PIT clock frequency is 10,485,760 Hz, giving a maximum delay of 409.6 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

Suppose that we want to:

- Make 60 changes of brightness
- In 2.5 sec

$$\text{Load value } = \frac{2.5 \,.\, 10{,}485{,}760}{60} - 1 = 436{,}906$$

6.3 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.

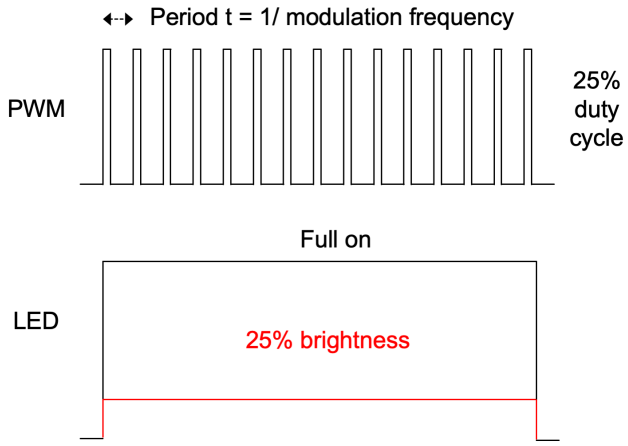
6.4 Interrupt Service Routine (ISR)

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 `main.c`: you need to remove the application specific part and replace it with you own code. As above, it is suggested you use a separate function.

7 Appendix B: Volume Control using PWM

7.1 Principles

The volume can be varied by using pulse width modulation (PWM). The principles of PWM are shown below. We require the modulation frequency $f = 1/t$ to be at least 200 Hz.



7.2 Modulation Frequency

The modulation frequency is given by:

$$\frac{\text{TPM clock frequency}}{\text{Prescale} \times \text{Modulo}}$$

7.3 Software API

The PWM software is in two files:

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

Four functions are provided:

1. `configureTPMClock()` which configures the clock to the TPM modules.
2. `configureTPM0forPWM()` and `configureTPM2forPWM()` which configures the TPM0 channel 1 and TPM 2 channels 0 and 1 for PWM output.
3. `setLEDBrightness(enum LED led, unsigned int brightness)`, which changes the brightness of one LED. The brightness is in the range 0 (off) to 31 (on full)

Further notes on this code:

1. The 3 pins used for the LEDs on the KL25Z development board cannot all be connected to the same TPM. We have to use module 0 for the blue LED and module 2 for red and green. The channels are also fixed by the pins.
2. Since the LEDs are wired active low, a high duty cycle makes the LED dim and vice versa.
3. The perceived brightness does not vary linearly with the duty cycle. A set of duty-cycle values, stored in an array, give a reasonable effect but it is not clear that the same values are best for the different colours.
4. Another provided function `configureLEDforPWM` sets the PCR (Pin Control Registers) to connect the three LED pins to the TPMs.