Practical 5

A template is provided which you must use.
We're going to develop and interrupt-driven system. You must use a timer to implement your timing processes . You should have no delay loops in your code. After initialising the timer, your code must sit in an infinite loop doing nothing until the timer ISR is called.

Part 1: (2)
TIM6 should generate an IRQ every 1 second. The ISR should cause the value on the LEDs to increment by 1. LEDs can start from 0.
You must use TIM6. The automaker will verify that TIM6 is running and generating interrupts.

Part 2: (2)
If SW0 is held, change the interrupt to be generated every 2 seconds.
Releasing SW0 should restore the timing back to a fixed 1 second.
It's okay if the timing change only happens on the next interrupt from when the button is interacted with.

Part 3: (3)
If SW1 is held, instead of incrementing the LEDs by 1 every time the timer fires, rather iterate through displaying the value in the PATTERNS block. By default: 0x00, 0x55, 0xAA, 0xBB, 0xCC, 0xDD, 0xEE, 0xFF. When it gets to the last value, it should loop around and display the first. The contents of this block will be modified at compile time so don't hard code values in.
The speed of the transitions should still be controlled by SW0. Ie: next pattern every 1 second is SW0 is not held, or 2 seconds if SW0 is held.
When SW1 is released, it should go back to the default behaviour of incrementing whatever is on the LEDs by one. It's not necessary to 'remember' what was on the LEDs last - just increment whatever value happens to be displayed.
It's not important which pattern is displayed first when SW1 is held, just so long as all patterns are looped through while the switch is held.

Bonus: (1)
Write the text "42" to the LCD screen.

Marked out of: 7
Available marks: 8

(see hints on next page)

Hints:

Don't even think about the later parts until you get your timer generating an interrupt. This can be finicky if it's your first time setting up interrupts!

You can use whatever ARR and PSC values you wish. Generally I find that setting the ARR and PSC to approximately the same values yields good dynamic range.
So, find approximately equal values for the ARR and PSC to solve part 1.
Then, for part 2, don't modify the PSC, just change the ARR.

All conditional logic should be contained in the ISR.
You have two blocks of logic in your ISR:
- a block which adjust the timing based on the state of SW0
- a block which sets whether the LEDs increment by one or cycle through the patterns based on SW1.

Part 3 may seem scary; it's not.
The suggested way to implement it to maintain an offset value which lives in memory (RAM) at some fixed address which you decide on. Every time the timer fires:

- fetch the offset from that fixed memory address, increment it by 1, wrapping it back to 0 if it's out of bounds.
- write the new offset back to memory so it's available next time the timer fires.
- fetch the pattern from the start of the PATTERNS block plus the offset.

The safest way to implement this is to give this offset value a fixed memory address. It's not safe to use the stack for storing data which you want accessible from interrupts. I hope you know why!