CMPE1250 – Characterizing the PIT – ICA #08

Introduction

If you have participated in the lecture(s) on the PIT, you are ready to create a library for the PIT that will provide reusable, simple functionality.

In addition to creating library functions, you will need to characterize the behavior of the PIT to gain a solid understanding of its benefits and limitations when used in polling mode.

A sample of the PIT header will be made available to you, to ensure that everyone uses the same prototypes. You are, as usual, permitted to create your own functions, and additional support functions as you see fit.

```
/// @brief Configures a channel (ch0 - ch3)
/// @param ch The channel in question (PIT_CH0 - PIT_CH3)
/// @param mt The micro-timer to be connected to (MT1 or MT0(default))
/// @param ie Enables or disables interrupt for the channel
void PIT_InitChannel(PIT_Channel ch, PIT_MicroTimer mt, PIT_Interrupt ie);

/// @brief Configures the channel to a 1[ms] event, fix connection to micro-timer1
/// @param ch The channel to be configured
void PIT_Set1msDelay(PIT_Channel ch);

/// @brief A blocking delay function in milliseconds
/// @param ch The channel to use with the delay
/// @param ms The number of milliseconds to delay
void PIT_Sleep(PIT_Channel ch, unsigned int ms);

/// @brief This enables the PIT. It is recommended to be called last.
///        Nothing will work if this is not called
void PIT_Start();
```

The **`PIT_InitChannel`** function is designed to configure the PIT module for the specified channel by enabling it, connecting to a micro-timer, and selecting the interrupt option. Your instructor will discuss the approach to doing this, and may provide some valuable code snippets.

The **`PIT_Set1msDelay`** function is designed to configure the PIT specified channel to generate a 1ms event. This will change depending on the bus speed, which will have to be passed in as a parameter or obtained through a getter function in the pll/clock library. This function will support the **`PIT_Sleep`** function and will have to be called at the beginning of it.

The **`PIT_Sleep`** function is designed to provide a blocking delay (the function won't return until the delay is complete). It will leverage the **`PIT_InitChannel`** and **`PIT_Set1msDelay`** functions to setup the specified channel for a 1ms delay, then repeatedly execute the 1ms delay, using polling, the specified number of times.

The **`PIT_Start`** function is designed to just enable the PIT peripheral. In addition to configuring the individual channel, the PIT module needs to be enabled for anything to work. Normally this operation is performed after everything has been configured.

Assignment

Because there is some complexity to these functions, you will need to do some comprehensive testing to ensure that they are working as intended.

Begin by creating a rough form of the `PIT_InitChannel` and `PIT_Start` implementation. In between those two calls, you will have to set up the time event by using, for instance, the formula:

$$N_{ticks} = BUS_{Speed}[MHz] \times Time_{Event}[\text{ms}]$$

Remember PIT Channel initialization sequence:

- Set `PITMUX`, `PITMTLDx`, `PITLDx` values.
- Enable/disable interrupts for the selected channel in `PITINTE`.
- Enable the channel in `PITCE`.

For enabling the PIT module:

- Enable the PIT module in `PITCFLMT`.

You should start my configuring a PIT channel to generate a 1[ms] event and test it using the AD2. After that, maybe try a different setting like a 100[ms] event. Again, test to ensure that this is functional.

Now, determine by hand calculation the values to fill out in the following table, if the values are rational for the given bus rate. If they are not, revise your code until the values are correct.

Remember: the tests at 8MHz and 20MHz exclude/include the Clock / PLL library call. Don't forget to comment in/out this line for tests at different bus rates or use another approach to test both bus speeds.

BUS Rate	PITMTLD0	PITLD0	Desired Interval	Calculated Interval
20MHz	39	49999	100ms	20E6 x 100E-3 = 2,000,000 = 40 x 50000
8MHz	15	49999	100ms	8E6 x 100E-3 = 800,000 = 16 x 50,000
20MHz	199	49999	500ms	20E6 x 500E-3 = 10,000,000 = 200 x 50,000
8MHz	75	49999	475ms	8E6 x 475E-3 = 3,800,000 = 76 x 50,000
20MHz	0	19999	1ms	20E6 x 1E-3 = 20,000
8MHz	0	55999	7ms	8E6 x 7E-3 = 56,000
20MHz	0	9999	500us	20E6 x 500E-6 = 10,000
8MHz	0	3839	480us	8E6 x 480E-6 = 3,840
20MHz	19	42999	43ms	20E6 x 43E-3 = 860,000 = 20 x 43,000
20MHz	9	24,689	12.345ms*	20E6 x 12.345E-3 = 246,900 = 10 X 24,690

*This interval will not necessarily be achievable unless you implement a 'best fit' algorithm. Consider this a special challenge for the aspiring programmers. You are not required to satisfy interval requests requiring non-simple factors (such things will not appear in assignments or exams).

To receive check-off for this assignment, you will need to demonstrate your code and measurements as directed by your instructor.