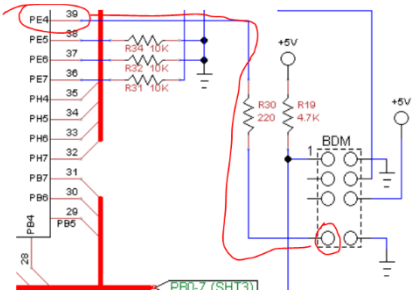
CMPE1250 – ICA #6, PLL/Clock Library and Proof of SPEED

You should know at this point the micro used for this course runs at 8[MHz], but how to verify that? There's a special pin where the clock output can be verified (PE4), which is routed to one of the pins on the debugger header:



Verify if you can see the clock output on the mentioned pin using a scope or the analog discovery. Such clock output could also be divided for displaying purposes, if the oscilloscope used has limited bandwidth. In order to divide the clock output, the EDIV0 and EDIV1 bits in the ECLKCTL register can be manipulated according to chapter 22 of the micro datasheet (22.3.2.13).

Built a new library (compilation unit)

After reviewing the documentation mentioned and the notes, implement the following function according to the header provided and Table 22-17 in the datasheet:

- *Clock_EnableOutput()*: Enable output specifying divider for it.

You are also required to provide an implementation to each of the remaining function prototypes.

- *Clock_Set8MHz():* Set clock to 8MHz (revert to default)
- *Clock_Set20MHz()*: Set clock to 20MHZ using PLL
- *Clock_Set24MHz():* Set clock to 24MHZ using PLL
- *Clock_GetBusSpeed():* Get current BUS speed.

For the function that gets the current BUS speed (getter), make sure you have a global unsigned long variable that keeps track of the BUS speed defined in *clock.c*. Such variable should be initialized to the default bus speed and then changed accordingly every time the clock speed is modified.

Part 1

Create a new project that blinks the red LED every 100ms (use technology from previous activities to set this up with a blocking delay). Do not include the PLL call in this part, as we want to establish a baseline rate.

Use your AD2 to capture the output on pin 82 and validate that your code is generating an approximate 5Hz square wave. Remember, the blocking delay needs to run twice to create a full wave, so the period of the output wave is two times the blocking delay period (one half the frequency).

Part 2

Now enable the *Clock_Set20MHz()* call in the one-time initializations section and run the modified code.

Do the same for Clock_Set24MHz().

For both options, complete the following table:

	Bus Frequency	**Blocking Delay Time**	**Frequency**
Baseline	8 MHz	100ms	5 Hz
Expected	20 MHz	40ms	12.5Hz
Measured with AD2	20 MHz	40ms	12.5Hz
Expected	24 MHz	33.33ms	15Hz
Measured with AD2	24 MHz	33.35ms	15Hz

A faster bus rate means faster instruction execution! We can now cram more instructions in per unit of time, and that is good if we need to do a lot over a short period of time. In the next micro course, you probably will!

Include captures of WaveForms showing the 8MHz, 20MHz and 24MHz waveforms, with included cursors/measurements, as discussed in class.

What is the per iteration time of your blocking delay (provide answer/evidence in code comments)?

Is it possible to run the micro with a bus rate of 40MHz? Why would or wouldn't you do this?

Explain you results in your Panopto video submission.