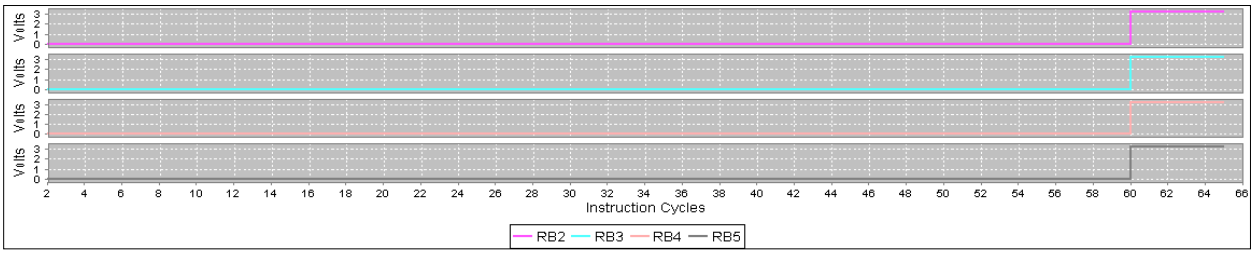
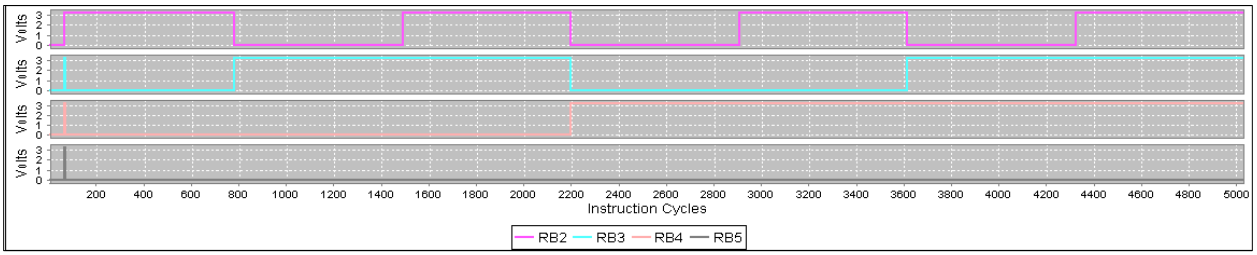
Screen shot of logic analyzer output of the simulation of the counter

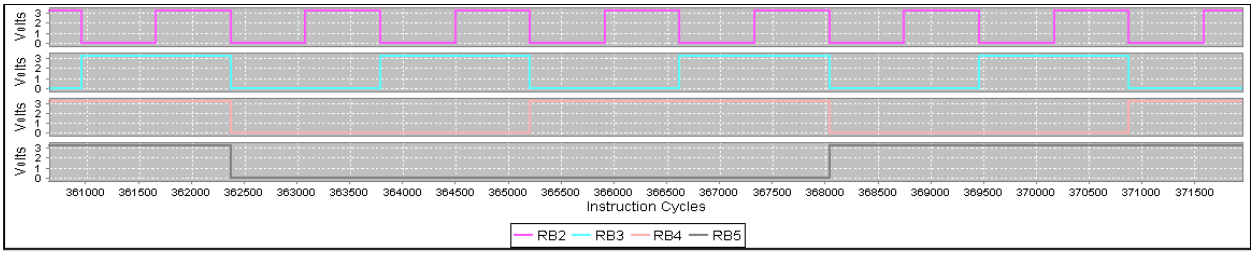
Counter halted at stopwatch cycle count 65 (4.0625 µs):



Counter halted at stopwatch cycle count 5052(



Counter halted at stopwatch cycle count 371950 (23.246875 ms)



I utilized the stopwatch debugging tool in MPLABX. This allowed me (when the breakpoint was set at each counter++ iteration) to see that it takes 709 cycles @ 44.3125 µs. Using the formula:

$$frequency = \frac{peroid\ events}{time\ frame\ of\ events} = \frac{709}{44.3125 * 10^{-6}} = 16{,}000{,}000 = 16MHz$$

Utilizing the debugger, formula, and stopwatch, I utilized the counter++ break point. I ran the simulation for 20 periods of RB2. The time it took to do that was 1.776562 ms, thus giving a frequency of 11.25769866 kHz. After being in the lab I was able to measure RB2 with an oscilloscope. The reading from the oscilloscope was a period of 88.99 µs and a measured frequency of approximately at 11.31 kHz. To narrow down the frequency even further I used the formula to get 11.23721766 kHz. From here it is easy to decipher the rest of the frequencies for the rest of the RBx pins. As we can see the simulation and hardware are very close in frequency.

Our light output is a representation of 16-bit binary number. In the code we shift our LATB output by two bits in-order to reserve RB0 and RB1 for input from the chipKit. Binary works in a simple way, by powers of 2. This can be translated to the viewing of the trigger event of an output pin being high. For example, since RB2 is our first bit in our 16-bit number it will be high for 2^0 and low for 2^0 continuing this pattern until we stop the program. Since RB3 is our bit in our 16-bit number it will be high for 2^1 and low for 2^1. RB4, high for 2^2 and low for 2^2. What does this mean for our frequency and how can we find the rest of the pin frequencies?

Once the RB2 frequency is measured, we can simply half that frequency and we have the frequency of the next pin. See table below:

Pin	Frequency (Hz)
RB2	11,258
RB3	5,629
RB4	2,815
RB5	1,407
RB6	703
RB7	352
RB8	176
RB9	88
RB10	44
RB11	22
RB12	11
RB13	5.5
RB14	2.75
RB15	1.375

Since RB15 is so slow we can see it and count it, I took the measurement of 28 led on events in 20 seconds. This gives us a frequency of 1.4 Hz! This is ideally in the range that was calculated in the table.

Using the logic analyzer, the stopwatch and a little math, we can see the simulation runs at 62.5 ns per cycle. Take 62.5 ns and multiply it by 16 MHz we get 1. Verifying the cycle speed of our clock in the simulation.

Our hardware measured on the oscilloscope and that in the simulation are slightly off. This is due to measuring tools and time to move through the bread board and into the oscilloscope. The clock is moving so fast that the oscilloscope in the lab couldn’t capture the correct data. Using a more precise instrument there’d be no doubt that the percent error would go way down.

This lab iterates a concept of psychophysics of vision, the flicker fusion rate. Ideally the human eye can perceive, while looking straight at something, a frequency of 60 Hz. This can be viewed with the RB10 pin (44 Hz). Moving the light jumper from RB10 to RB9 it becomes a solid light. This is because RB9 is measured 88 Hz. Flicker fusion is an important idea in imaging technologies moving forward. If you move the bread board at 88 Hz you can perceive the flicker of the light. It is said in the Flicker Fusion principal that most people can't detect the moving flicker at 400 Hz. Testing the theory is easily done with the RB7 and RB6 pins (moving it side to side) and the theory holds.

The MCU can do 16 million operations in 1 second, hence 16 MHz. With it taking 62.5 ns per cycle and 709 cycles to execute our loop back to break. It takes 44.3125 µs to execute our instruction cycle. This means our computer can do about 22,567 instruction cycles in 1 second.