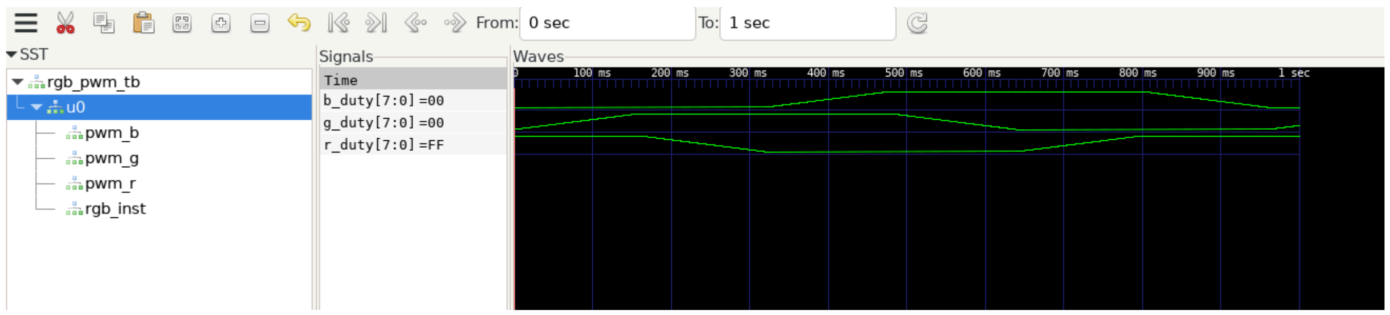
MiniProject 2

For this project, I implemented the RGB LED controller using a finite state machine approach. The idea is simple: the full HSV color wheel is split into six segments - red to yellow, yellow to green, green to cyan, cyan to blue, blue to magenta, and magenta to red. Each segment corresponds to one "state" in the FSM. As the LED cycles through the colors, the FSM transitions from one state to the next when the progress variable reaches its maximum value, ensuring a smooth and predictable sequence through the color wheel. Using an FSM makes it easy to reason about the color transitions and avoids having to deal with complex calculations for hue in each clock cycle.

Within each state, the brightness of the individual RGB components is controlled by an 8-bit progress counter. This counter increases steadily from 0 to 255 over the course of each segment. The current value of the progress counter directly determines the duty cycle for each RGB LED, so that, for example, when transitioning from red to yellow, the red LED stays at full brightness while the green LED ramps up from 0 to 255. This way, the LEDs smoothly blend between colors, producing the continuous gradients we expect from the HSV wheel. To get exactly one second per full cycle, I implemented tick splitting: the progress counter increments once every 7812.5 ticks of the 12 MHz clock. Since we can't represent half-ticks, it alternates between 7812 and 7813 ticks for each increment using a toggle bit. This keeps the timing exact over the full cycle while maintaining smooth ramps for each color.

The PWM module takes the 8-bit duty cycles generated by the FSM and converts them into active-low PWM signals for the RGB LED. In simulation, the duty cycles themselves can be visualized in GTKWave, showing smooth linear ramps corresponding to each segment of the color wheel. The combination of the FSM for color states, the progress counter, and the tick splitting ensures that the RGB LED smoothly cycles through all hues over exactly one second on the simulation, just like it does on the physical board.



```
rgb_pwm_tb
  u0
    pwm_b
    pwm_g
    pwm_r
    rgb_inst
```

```
Time
b_duty[7:0] =00
g_duty[7:0] =00
r_duty[7:0] =FF
```

Note that the pwm module does correctly assume active low for the led pins, while the duty cycle logic wires represent the percentage brightness - the brightness goes up with the duty cycle.