Andrew Friedman
Lab 5 Report
ECE 2031 CS
23 June 2023

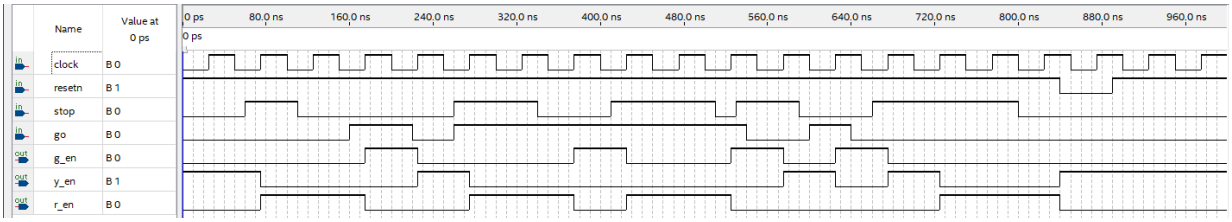


**Figure 1.** Simulation of the state machine implemented using VHDL, highlighting the transitions between states in response to various input test waveforms.

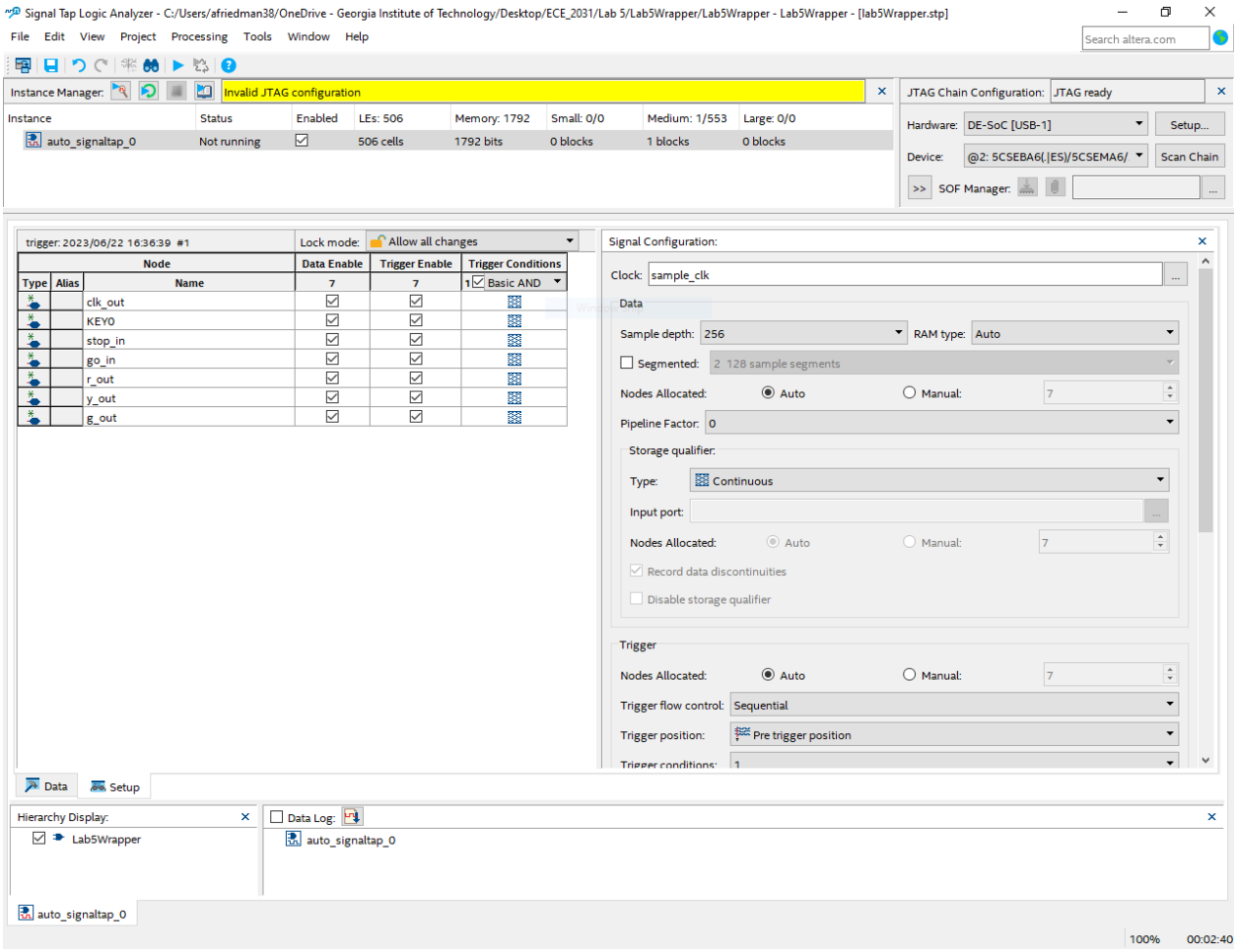


**Figure 2.** Screenshot of the fully configured Signal Tap Logic Analyzer window, showcasing the selected signal nodes for monitoring, the chosen 20 Hz sample clock, an acquisition length of 256 samples, and acceptance of default settings for continuous acquisition and simplest sampling.

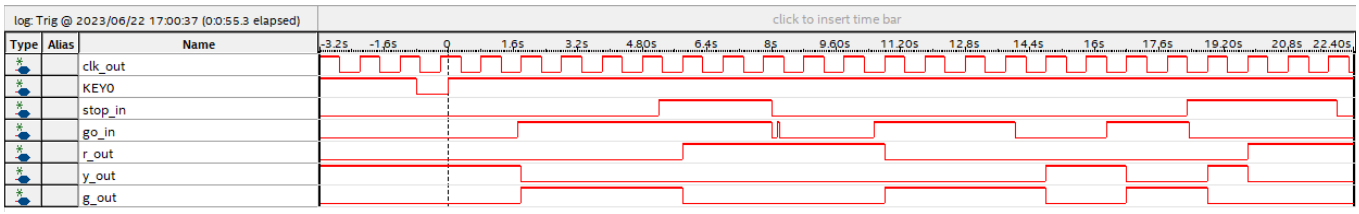

**Figure 3.** Signal Tap acquisition displaying the successful execution of specified event sequence, with the correctly adjusted time scale presented in seconds.

# APPENDIX A
# VHDL STATE MACHINE CODE IMPLEMENTING STATUS LIGHTING

```
-- PRELAB5.VHD
-- Three-State Moore State Machine For Status Lighting
-- Andrew Friedman
-- ECE 2031 CS
-- 06/23/2023

library ieee;
use ieee.std_logic_1164.all;

entity PreLab5 is

	port(
		stop	: in	std_logic;
		go	: in	std_logic;
		clock	: in	std_logic;
		resetn	: in	std_logic;
		r_en	: out	std_logic;
		y_en	: out	std_logic;
		g_en	: out	std_logic
	);

end entity;

architecture rtl of PreLab5 is

	-- Build an enumerated type for the state machine
	type state_type is (r_light, y_light, g_light);

	-- Register to hold the current state
	signal state   : state_type;

begin

	-- Logic to advance to the next state
	process (clock, resetn)
	begin
		if resetn = '0' then
			state <= y_light;
		elsif (rising_edge(clock)) then
			case state is
				when r_light =>
					if stop = '0' and go = '0' then
						state <= r_light;
					elsif stop = '0' and go = '1' then
```

```
                                        state <= g_light;
                                elsif stop = '1' and go = '0' then
                                        state <= r_light;
                                elsif stop = '1' and go = '1' then
                                        state <= r_light;
                                end if;
                        when y_light =>
                                if stop = '0' and go = '0' then
                                        state <= y_light;
                                elsif stop = '0' and go = '1' then
                                        state <= g_light;
                                elsif stop = '1' and go = '0' then
                                        state <= r_light;
                                elsif stop = '1' and go = '1' then
                                        state <= r_light;
                                end if;
                        when g_light =>
                                if stop = '0' and go = '0' then
                                        state <= y_light;
                                elsif stop = '0' and go = '1' then
                                        state <= g_light;
                                elsif stop = '1' and go = '0' then
                                        state <= y_light;
                                elsif stop = '1' and go = '1' then
                                        state <= r_light;
                                end if;
                end case;
        end if;
end process;

-- Output depends solely on the current state
process (state)
begin
        case state is
                when r_light =>
                        r_en <= '1';
                        y_en <= '0';
                        g_en <= '0';
                when y_light =>
                        r_en <= '0';
                        y_en <= '1';
                        g_en <= '0';
                when g_light =>
                        r_en <= '0';
```

```
                                    y_en <= '0';
                                    g_en <= '1';
                end case;
        end process;

end rtl;
```