Module	EE6621 (ASICS1)
Date	13/12/2024
Assignment No	Coursework Project (pg03)
Student Name and ID	Luke Griffin 21334528

A project report with a concise description of your design.

- My design was expanded by using rt2 as a starting template.
- I implemented the display to produce the expected outputs.
- Next, I made a module and testbench for the signal_out value
- I extended this to allow its tph_value to be configurable and added to the display with a constant frequency of 50.0kHz.
- In fsm_game I added new states and made a variable tph_value that could be changed using the the 2 fpga buttons by incrementing its value.
- Next, I got the pushbuttons working using the test02 example code and assigned the values to change states and tph_value
- I got the onboard ja pins working so we could verify that the signal_out tph value was changing when the user configured it
- I enabled the buzzer to work and concluded that I was happy with my M Project implementation
- Next I moved onto the large and started on the PRBS random generator creating a module and assigning it to the signal_out value when a certain state is reached
- The configurable frequency was then implemented in a similar way to tph_output except using a search algorithm to compute the cycle number based off the frequency value e.g 50.0kHz = 2000 cycles
- Finally, we attempted to start on the composite video but only being able to print ‘pg03’ on the screen. (had to be removed as it was affecting out tph value)

Was any section of your report or HDL code AI-generated? -> no

List of implemented (and missing) features.

Features (M)	Implemented? (yes/no)
signal_out and signal_cycle signals	yes
Configurable tph value (0.1us -> 9.9us)	yes
Buttons working (muxpb and buttons[1:0])	Yes (ESC button was working but seemed to stop – not a hardware problem)
7-segment display with correct start-mode info	yes
Pulse-mode with correct defaults (2.5us, 50Hz)	yes
Blinking state when editing tph values	yes

Buzzer when any button pressed	yes
Pulse signal accessible through Pmod Port	yes

Features (L)	Implemented? (yes/no)
User Adjustable Pulse Repetition Frequency	yes
PBRS-mode	yes

Features (XL)	Implemented? (yes/no)
Composite Video (name)	Partially (pg03 being displayed on screen)
Composite Video (data)	no

In your lab report, explain in exact detail how you approached and solved this challenge. Add a postimplementation hardware utilization report to your project report.

- Taking the equation u hinted into account:

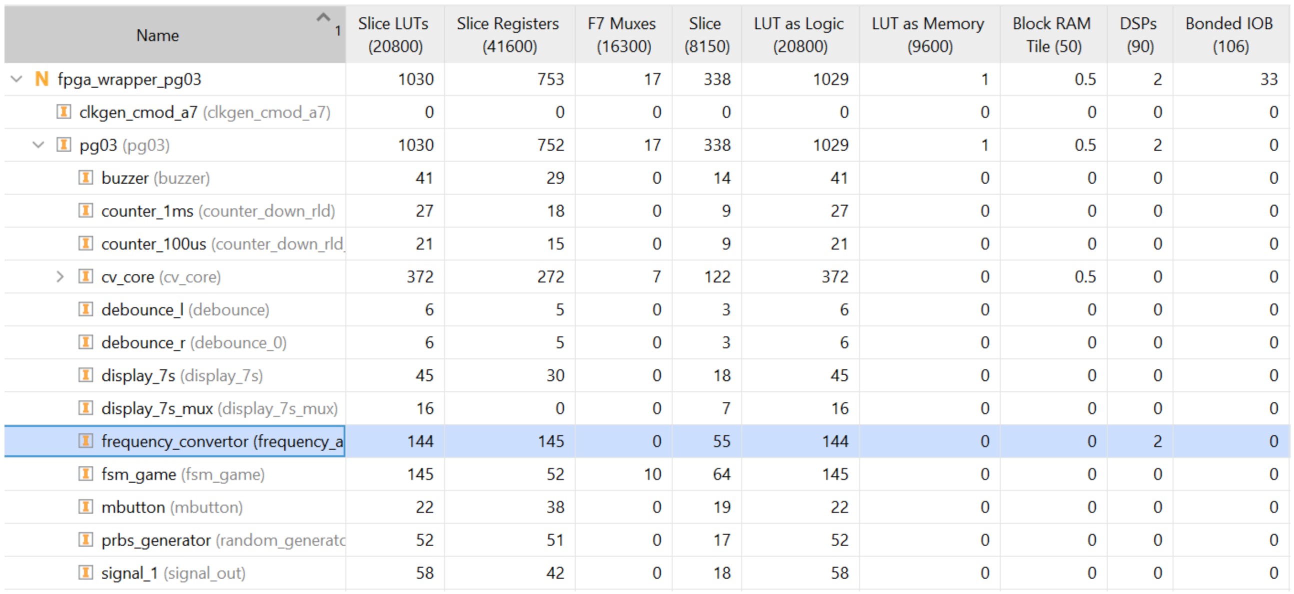
$$\frac{1}{f} = T \quad \Leftrightarrow \quad f * T = 1$$

- Also, u hinted that it doesn't need to be done straight away
- Instead of doing the division I performed a binary search to find the pulse period for a given frequency, utilizing simple arithmetic-add, shift, compare-along with a multiplier to converge quickly on the proper period given f * T = constant.
- It maintains upper and lower bounds, calculates the midpoint, and compares the product against a target value to reduce the search range.
- It repeats this after many iterations to return to a final period.
- Major hardware utilization includes basic logic LUTs/FFs and a DSP block utilized in the multiplication, hence giving a very good balance between simplicity and resource efficiency.

Analyse and critique your hardware utilization.

- The total design uses 1,030 Slice LUTs (out of 20,800 available) and 753 Slice Registers (out of 41,600 available).
- Frequency Converter (frequency_algorithm) consumes 144 LUTs and 145 Registers, making up approximately 14% of total LUT usage and 19% of register usage.
- This indicates that the frequency conversion block is a significant contributor to overall resource consumption.

While 14% of LUT usage is reasonable, there may still be room to optimize the frequency_algorithm module by reducing its resource usage further, potentially through shared resources or simplified logic.



Name	Slice LUTs (20800)	Slice Registers (41600)	F7 Muxes (16300)	Slice (8150)	LUT as Logic (20800)	LUT as Memory (9600)	Block RAM Tile (50)	DSPs (90)	Bonded IOB (106)
fpga_wrapper_pg03	1030	753	17	338	1029	1	0.5	2	33
clkgen_cmod_a7 (clkgen_cmod_a7)	0	0	0	0	0	0	0	0	0
pg03 (pg03)	1030	752	17	338	1029	1	0.5	2	0
buzzer (buzzer)	41	29	0	14	41	0	0	0	0
counter_1ms (counter_down_rld)	27	18	0	9	27	0	0	0	0
counter_100us (counter_down_rld	21	15	0	9	21	0	0	0	0
cv_core (cv_core)	372	272	7	122	372	0	0.5	0	0
debounce_l (debounce)	6	5	0	3	6	0	0	0	0
debounce_r (debounce_0)	6	5	0	3	6	0	0	0	0
display_7s (display_7s)	45	30	0	18	45	0	0	0	0
display_7s_mux (display_7s_mux)	16	0	0	7	16	0	0	0	0
frequency_convertor (frequency_a	144	145	0	55	144	0	0	2	0
fsm_game (fsm_game)	145	52	10	64	145	0	0	0	0
mbutton (mbutton)	22	38	0	19	22	0	0	0	0
prbs_generator (random_generatc	52	51	0	17	52	0	0	0	0
signal_1 (signal_out)	58	42	0	18	58	0	0	0	0

Figure 0: Hardware Utilization report

Do Vivado synthesis or implementation runs produce errors or warnings? Analyse them, and briefly discuss in your project report.

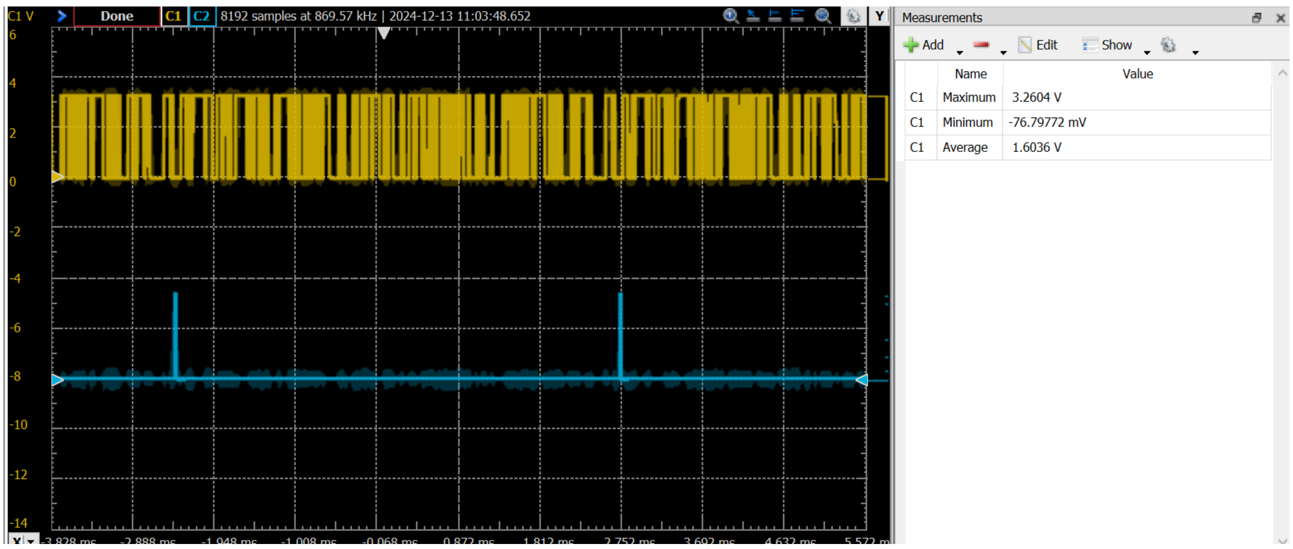
Yes, it produces warnings, they can all be ignored all apart from the 1 critical warning for timing.

Provide details of your design and implementation in your project report.

- The PRBS generator here gives an 8-bit LFSR output whose output frequency is to be set by the user.
- The LFSR here is designed to utilize XNOR-based feedback taps at the 8th, 6th, 5th, and 4th stages for a maximum length sequence of 256 different states with no lock-up conditions.
- This is constantly going through those states at the selected rate, producing a PRBS signal, along with the forward advancement of a rollover signal every 256 sequences for the purpose of synchronization.

Describe your bench verification approach. Average the output generates an equal number of ones and zeros.

- By observing the bitstream on a scope and measuring its average value with the analog discovery, it's straightforward to confirm that, on average, the Voltage is 1.6V which is half of 3.2V which indicates half are high and half are low.



	Name	Value
C1	Maximum	3.2604 V
C1	Minimum	-76.79772 mV
C1	Average	1.6036 V

Figure X: Average Measurement of PRBS mode

List of re-used and adapted building blocks from rt2 or test02.

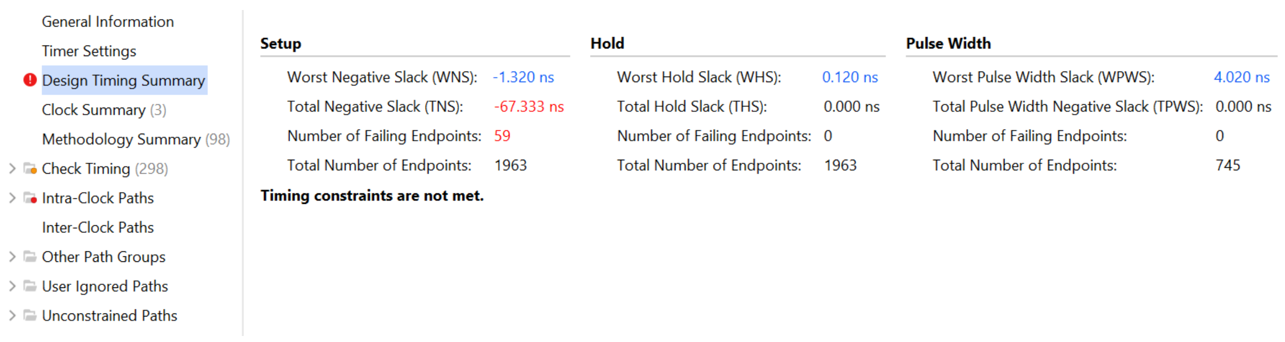
Modules re-used are –> buzzer.v, cllkgen_cmod_a7.v, counter_down_rld.v, debounce.v, display_7s.v, display_7s_mux.v, mbutton.v, synchroniser_3s.v, cv_char, cv_charrom_access, cv_charrom, cv_control

Modules edited and reused -> fpga_wrapper_pg03.v (fpga_wrapper_rt2.v), fsm_game.v, rt2.v (pg03.v), cv_core

Includes reused -> fsm_game__states.v, timing.v, wordlength.v

Constraint files reused -> cmod-a7-config.xdc, rt2.xdc

Vivado implementation reports (timing, clock, device utilisation).



Setup

Worst Negative Slack (WNS):	-1.320 ns
Total Negative Slack (TNS):	-67.333 ns
Number of Failing Endpoints:	59
Total Number of Endpoints:	1963

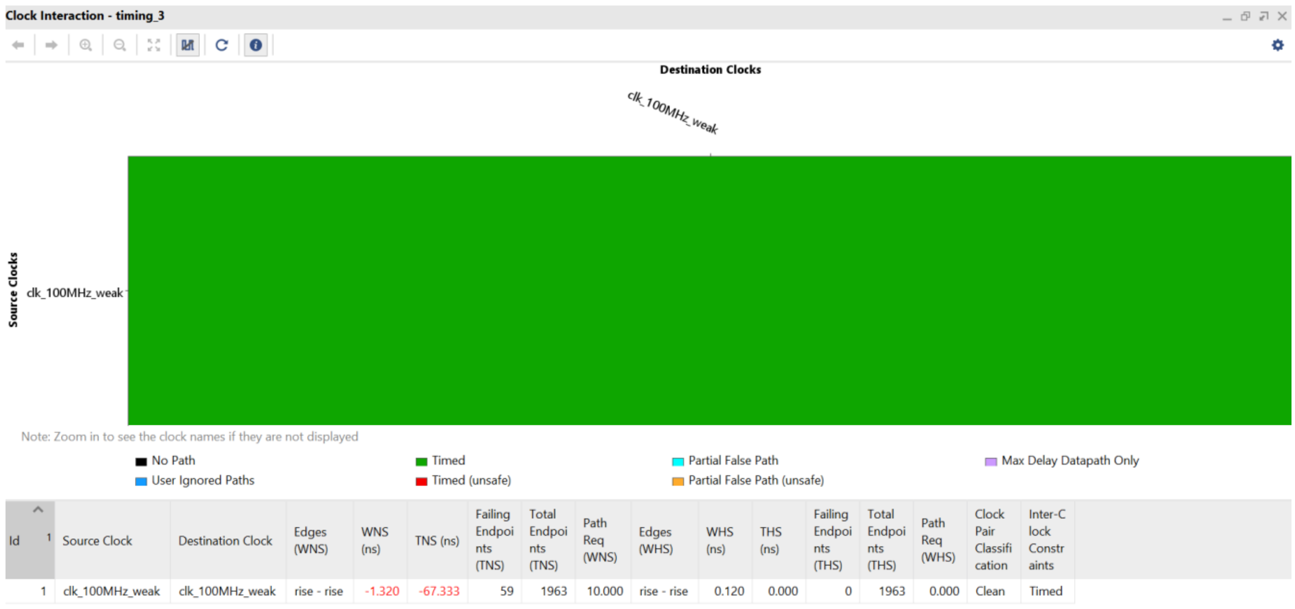
Hold

Worst Hold Slack (WHS):	0.120 ns
Total Hold Slack (THS):	0.000 ns
Number of Failing Endpoints:	0
Total Number of Endpoints:	1963

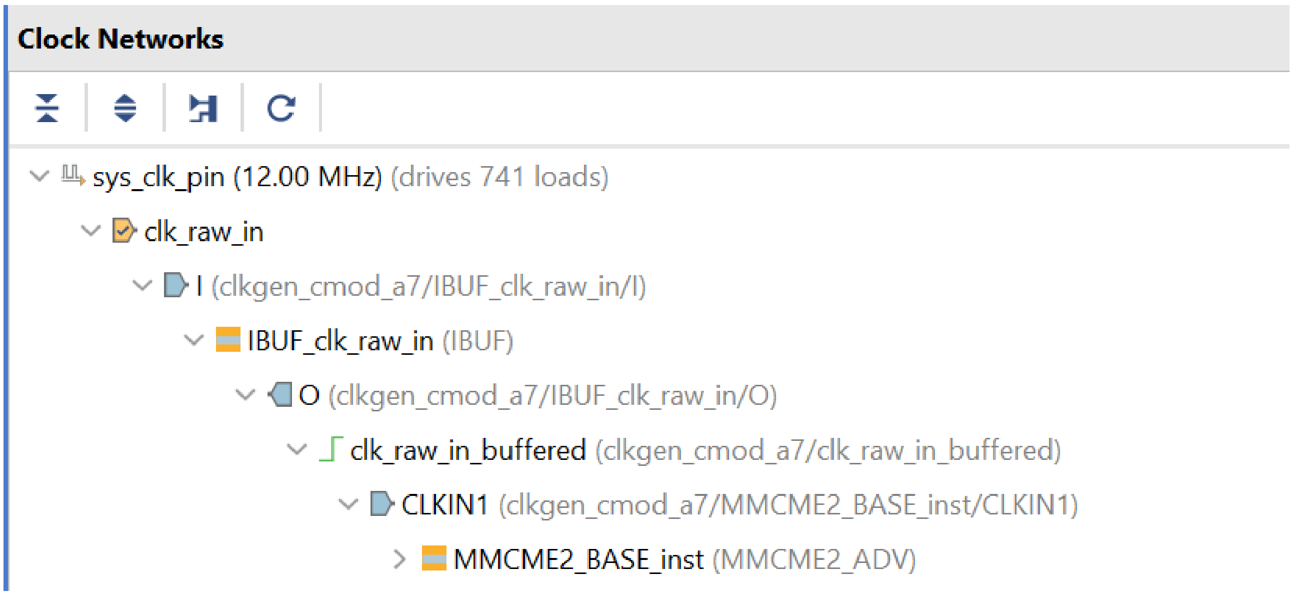
Pulse Width

Worst Pulse Width Slack (WPWS):	4.020 ns
Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0
Total Number of Endpoints:	745

Timing constraints are not met.



Id	Source Clock	Destination Clock	Edges (WNS)	WNS (ns)	TNS (ns)	Failing Endpoints (TNS)	Total Endpoints (TNS)	Path Req (WNS)	Edges (WHS)	WHS (ns)	THS (ns)	Failing Endpoints (THS)	Total Endpoints (THS)	Path Req (WHS)	Clock Pair Classification	Inter-Clock Constraints
1	clk_100MHz_weak	clk_100MHz_weak	rise - rise	-1.320	-67.333	59	1963	10.000	rise - rise	0.120	0.000	0	1963	0.000	Clean	Timed


Clock Networks

```
sys_clk_pin (12.00 MHz) (drives 741 loads)
  clk_raw_in
    I (clkgen_cmod_a7/IBUF_clk_raw_in/I)
      IBUF_clk_raw_in (IBUF)
        O (clkgen_cmod_a7/IBUF_clk_raw_in/O)
          clk_raw_in_buffered (clkgen_cmod_a7/clk_raw_in_buffered)
            CLKIN1 (clkgen_cmod_a7/MMCME2_BASE_inst/CLKIN1)
              MMCME2_BASE_inst (MMCME2_ADV)
```

Measurement results (scope plots), showing the produced output signals in various operating conditions. Document your bench verification efforts using scope measurements. Add scope plots at default operation (TPH=2.5 µs, f=50.0 kHz), and extreme corners of operation (e.g. TPH=9.9 us, f=99.9 kHz).

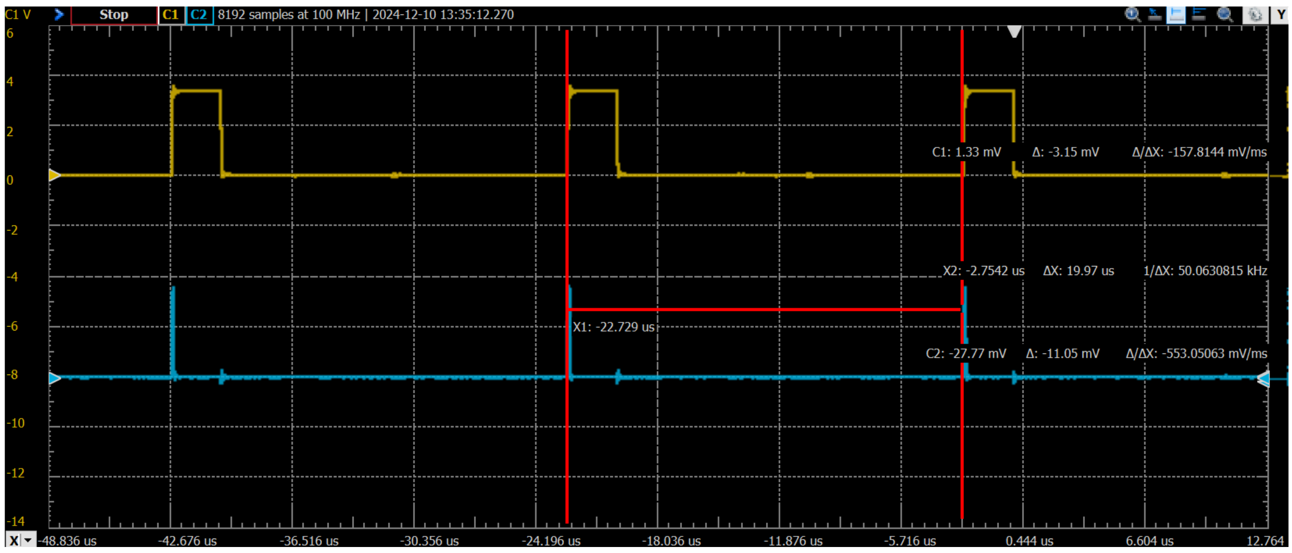

Stop
C1
C2
8192 samples at 100 MHz | 2024-12-10 13:35:12.270
C1: 1.33 mV Δ: -3.15 mV Δ/ΔX: -157.8144 mV/ms
X2: -2.7542 us ΔX: 19.97 us 1/ΔX: 50.0630815 kHz
X1: -22.729 us
C2: -27.77 mV Δ: -11.05 mV Δ/ΔX: -553.05063 mV/ms

Figure 1: Initial Operating conditions (tph = 2.5 µs *& f = 50Hz)*

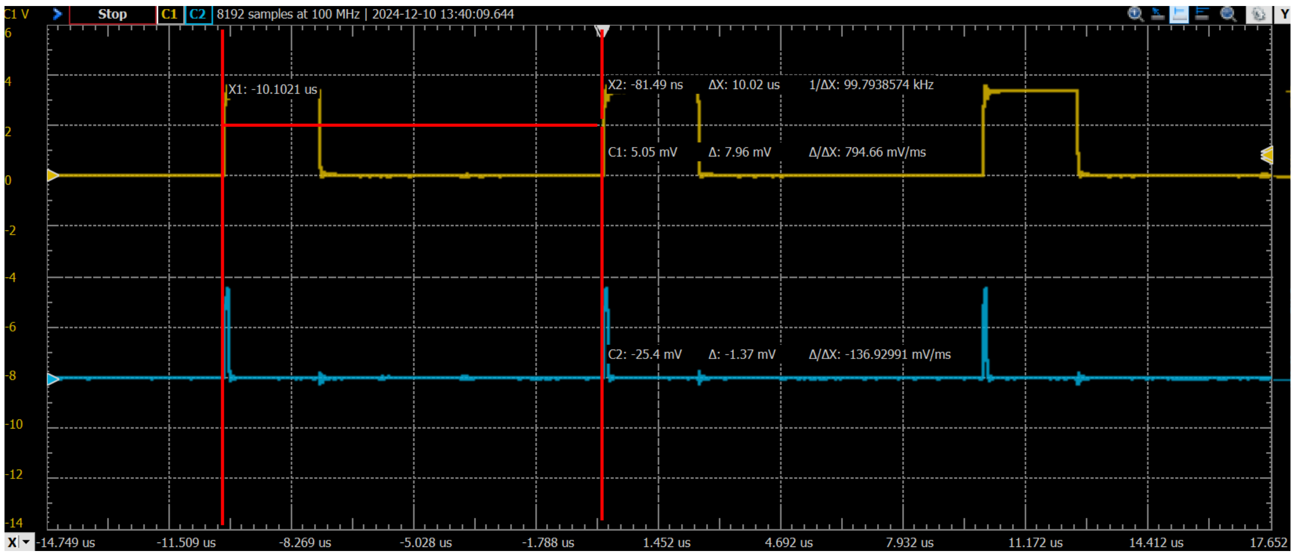

Stop
C1
C2
8192 samples at 100 MHz | 2024-12-10 13:40:09.644
X1: -10.1021 us
X2: -81.49 ns ΔX: 10.02 us 1/ΔX: 99.7938574 kHz
C1: 5.05 mV Δ: 7.96 mV Δ/ΔX: 794.66 mV/ms
C2: -25.4 mV Δ: -1.37 mV Δ/ΔX: -136.92991 mV/ms

Figure 2: Maximum Operating conditions (tph = 9.9 µs *& f = 99.9Hz)*

99KHz on screen as was closest I could get using the measurements tool

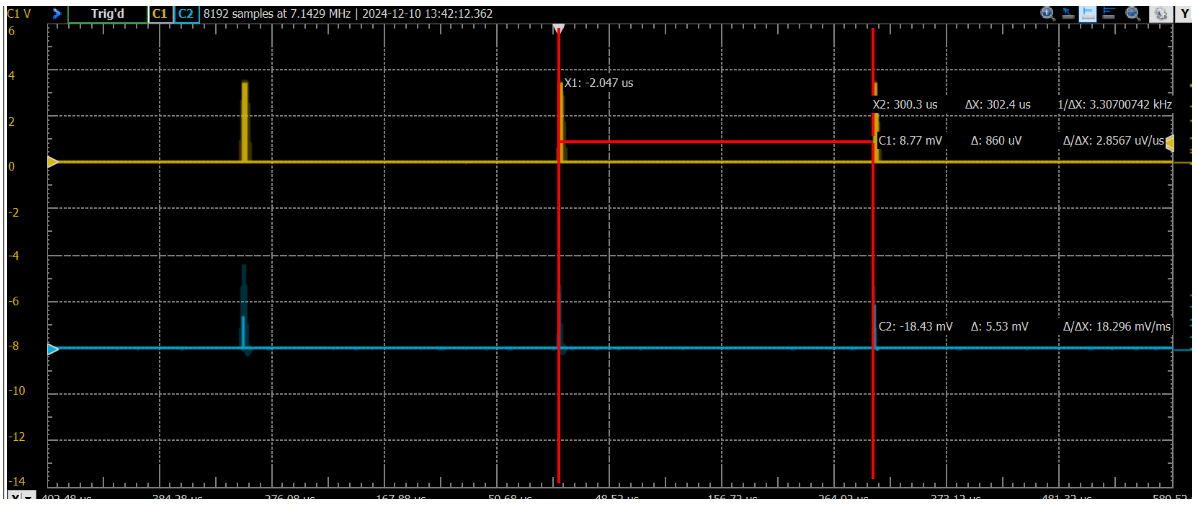


Figure 3: Minimum Operating conditions (tph = 0.1 µs & f = 3.2Hz)

3.3KHz on screen as was closest I could get using the measurements tool

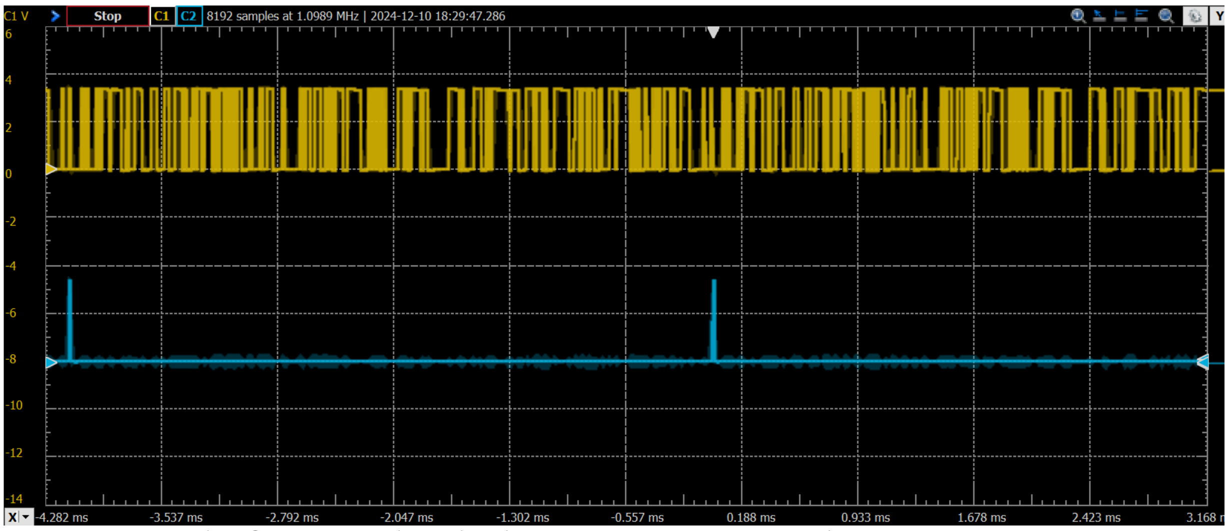


Figure 4: Example of scenario when the device is put into PRBS mode.

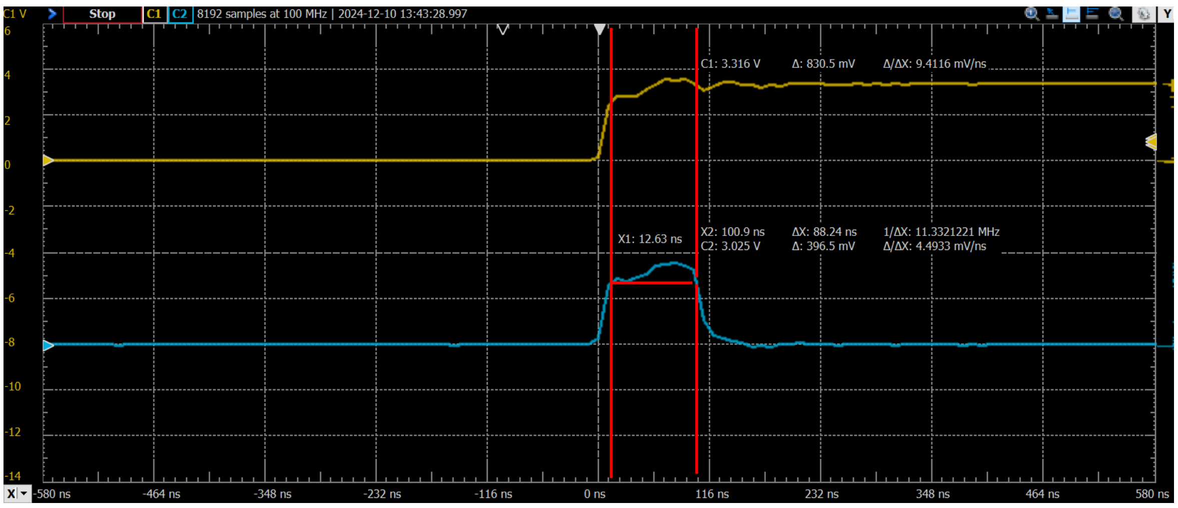
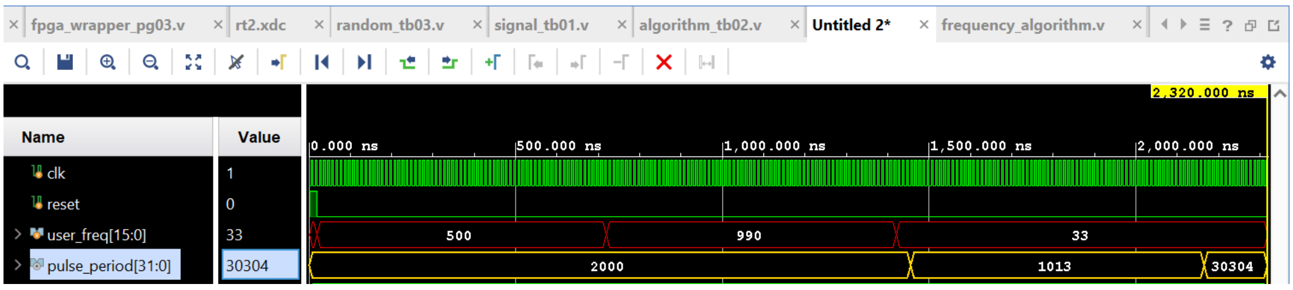


Figure 4: Measurement of signal_cycle showing that the pulse width is 100ns



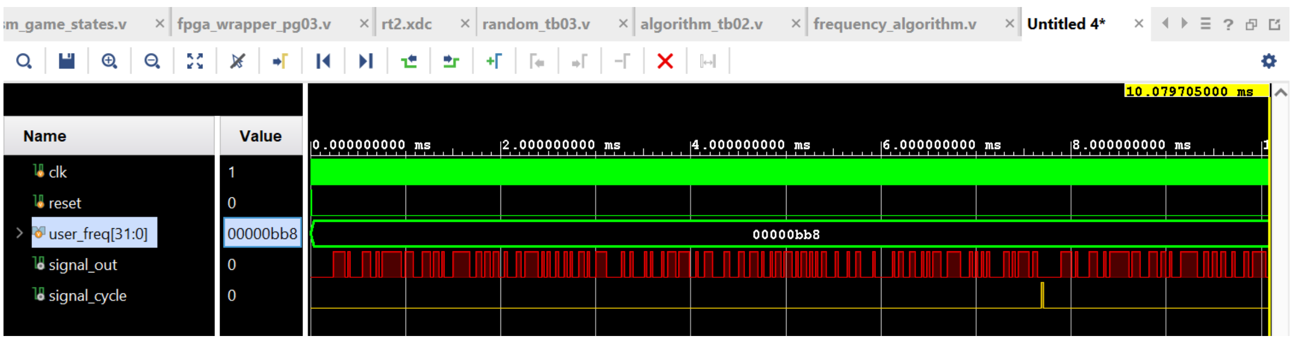


Figure 5: Testbench Simulation results for frequency_algorithm and random_generator

Issues encountered:

My ESC button was functioning throughout the project until I was cleaning up my code and it stopped. Couldn't find the issue when looking back on it. This could have been prevented if I used git

Video output only produce 4 letters and no more. Didn't have time to find a solution. Probable solution is somethings length isn't initialised long enough.

Only have testbenches for the frequency_algorithm and random_generator as I was writing my new code into the previous testbenches. Also, could have been prevented if git was used.

Acknowledgment: Taha Al-Salihi and Michael Cronin supported with problems encountered.