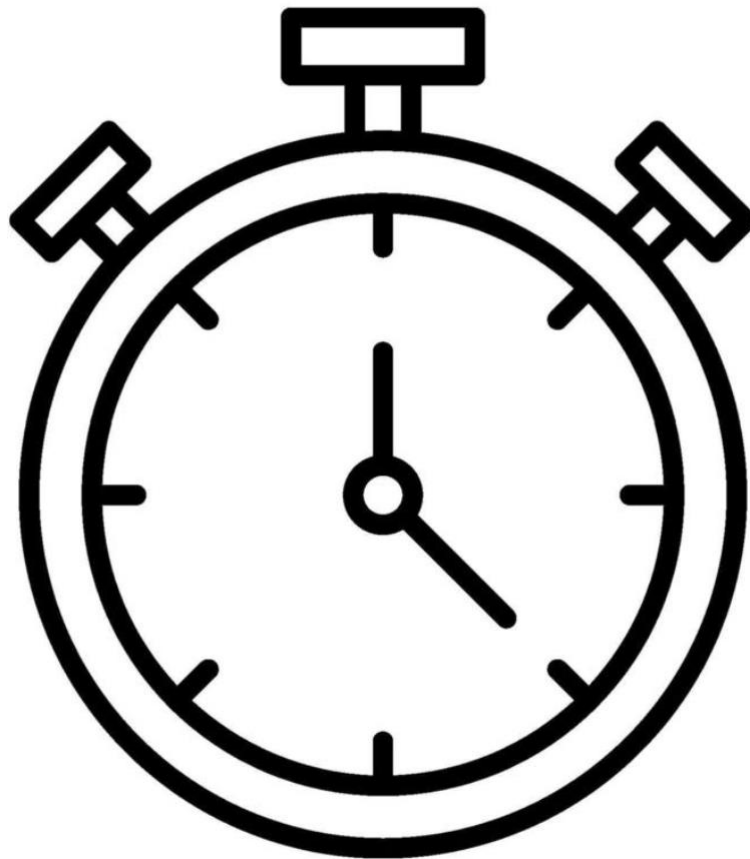


**FPGA-BASED DIGITAL SYSTEM DESIGN**  
**COURSEWORK**  
**HUMAN RESPONSE TIMER**



Mohammad Ali Hassan (13077008)

# TABLE OF CONTENTS

## SECTION I

### DESIGN

1.1 INTRODUCTION.....	1
1.2 DESIGN .....	1
1.3 MODELLING OF THE SYSTEMS.....	2

### DESIGN IMPLEMENTATION

1.4 IMPLEMENTATION .....	4
1.4.1 CLOCK DIVIDER.....	4
1.4.2 SEVEN SEGMENT DISPLAY .....	5
1.4.3 RANDOM GENERATOR BLOCK .....	6
1.4.4 LATCH .....	6
1.4.5 FOUR-BIT COMPARATOR .....	7
1.4.6 FOUR-BIT COUNTER .....	7
1.4.7 COUNTER-MAPPING .....	7
1.4.8 COMPARATOR INTERFACE.....	8
1.4.9 BUTTON DEBOUNCE.....	8
1.4.10 REACTION TIME.....	9
1.4.11 POWER CONSUMPTION & RESOURCE UTILIZATION.....	10

### SIMULATION AND TEST BENCH

1.5 SIMULATION .....	11
1.5.1 CLOCK DIVIDER TEST BENCH .....	11
1.5.2 RANDOM NUMBER GENERATOR TEST BENCH.....	11
1.5.3 LATCH TEST BENCH .....	12
1.5.4 COUNTER TEST BENCH .....	12
1.5.5 COMPARATOR TEST BENCH .....	13
1.5.6 REACTION TIME TEST BENCH .....	13
1.5.7 SEVEN SEGMENT DISPLAY TEST BENCH .....	14
1.5.8 BUTTON DEBOUNCE TEST BENCH .....	15

### HARDWARE IMPLEMENTATION

1.6 GROUP DEMOSTRATION .....	16
------------------------------	----

### CONCLUSION

1.7 CONCLUSION .....	16
----------------------	----

## SECTION II

### APPENDIX

#### MAIN BLOCK VHDL CODE

2.1.1 CLOCK DIVIDER.....	17
2.1.2 SEVEN SEGMENT DISPLAY.....	19
2.1.3 RANDON GENERATOR BLOCK .....	23
2.1.4 LATCH.....	24
2.1.5 FOUR-BIT COMPARATOR.....	26
2.1.6 FOUR-BIT COUNTER.....	27
2.1.7 COUNTER MAPPING .....	28
2.1.8 COMPARATOR INTERFACE .....	29
2.1.9 BUTTON DEBOUNCE .....	31
2.1.10 REACTION TIME .....	32
2.1.11 HUMAN REACTION (MAIN).....	34

#### TEST BENCH VHDL CODE

2.2.1 CLOCK DIVIDER TEST BENCH .....	42
2.2.2 4-BIT COMPARATOR TEST BENCH .....	43
2.2.3 4-BIT COUNTER TEST BENCH.....	46
2.2.4 SEVEN SEGMENT DISPLAY TEST BENCH .....	48
2.2.5 LATCH TEST BENCH .....	51
2.2.6 RANDOM NUMBER GENERATOR TEST BENCH.....	53
2.2.7 BUTTON DEBOUNCE TEST BENCH.....	54
2.2.8 REACTION TIME TEST BENCH .....	56

## SECTION I

### 1.1 INTRODUCTION

This report presents the development and implementation of a timer circuit designed to measure human response time following exposure to a visual stimulus. The timer circuit was executed on a Nexys-4DDR board, and its algorithm was created using VHDL coding. The system design is structured into ten distinct sections, each focusing on a specific aspect of the design. These sections were developed independently and later integrated into the main final section. To ensure accuracy and reliability, comprehensive test benches were employed to simulate the circuit's output and identify any potential errors. By following this organized approach, the report provides a clear and systematic overview of the human reaction timer circuit's design and its successful implementation on the hardware platform. The circuit serves as a valuable tool for studying cognitive responses, and its design, implementation, testing, and performance analysis are covered in the report.

### 1.2 DESIGN

The circuit features three input pushbuttons, a discrete LED as the stimulus, and a seven-segment LED display for presenting relevant information. The initial state displays a welcoming message, "HI," and activates the test when the start button is pressed. Following a random interval of 2 to 15 seconds, the stimulus LED is activated, and the timer starts counting milliseconds, which are displayed on the seven-segment LED. The response time is displayed once the stop button is pressed, typically falling within the range of 150 to 300 milliseconds for most individuals. In cases where the stop button is not pressed, the timer halts after 1 second, showing "1000" on the seven-segment LED. If the stop button is pressed before the stimulus LED activates, the circuit displays "99999999" and terminates the measurement process.

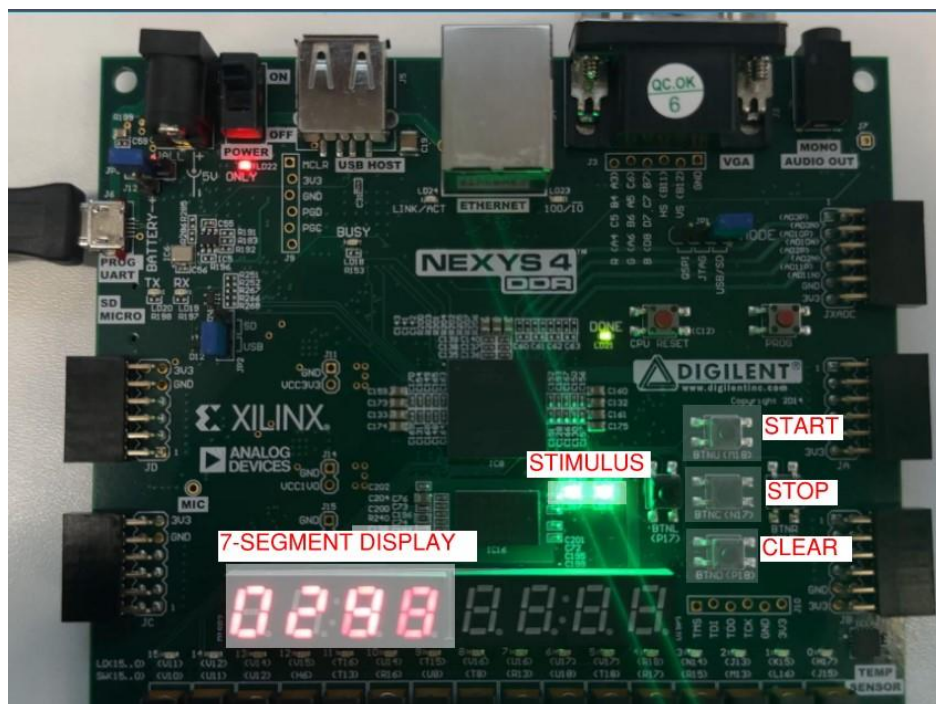


Fig: Circuit board

The process algorithm is described by a flowchart and is shown below:

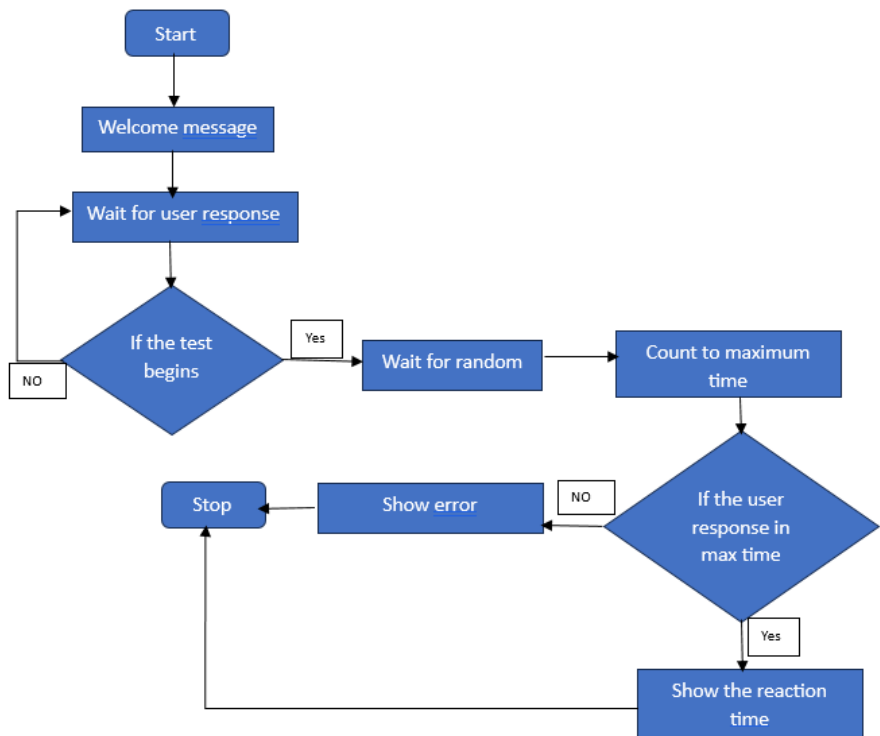


Fig: System flowchart

1.3 MODELLING OF THE SYSTEMS

Prior to implementation, it is critical to create a model of the system. A state machine diagram is used to model the system, which serves as a graphical depiction of the system's behaviour across time. This graphic depicts the numerous states that the system can adopt as well as the transitions that occur in reaction to external events or actions. The state diagram below depicts the system's operating modes, interactions, and decision-making processes in an organised and visible manner. Designers may effectively analyse the system's reaction to various inputs or stimuli using this modelling method. Furthermore, state machine diagrams serve in requirement analysis, risk minimization, and effective stakeholder communication. They also act as useful documentation, making future system maintenance and improvements easier. The state machine diagram emphasises the importance of system modelling by contributing to a well-designed and understandable system, hence expediting the implementation process.

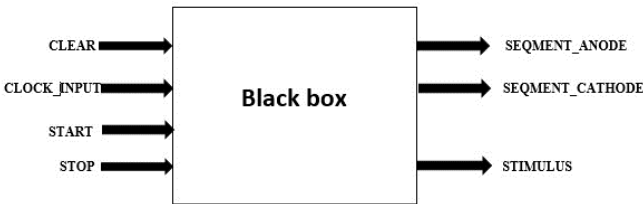


Fig: Model black box

The system diagram reveals five distinct states: 'wait for input', 'wait for random time', 'count up', 'error', and 'stop and display'. Upon pressing the button ('start = 1'), the system transitions to the 'wait for random time' state. Subsequently, if an input is detected, the system proceeds to the 'count up' state, with the trigger limits set to 1. In case no input is received, the system enters the 'error' state, leading to system termination. From the 'error' state, the system can return to the 'wait for input' state upon detecting a 'clear = 1' signal. When in the 'count up' state, the system either moves back to 'wait for input' upon 'clear = 1', or the counter continues counting time until the stop button is pressed, prompting the display. After reaching the 'stop and display' state, a 'clear = 1' signal from the clear button triggers a transition back to the 'wait for input' state, completing the cycle. The diagram provides a clear representation of the system's state transitions and the conditions for each change, facilitating a comprehensive understanding of the system's behaviour.

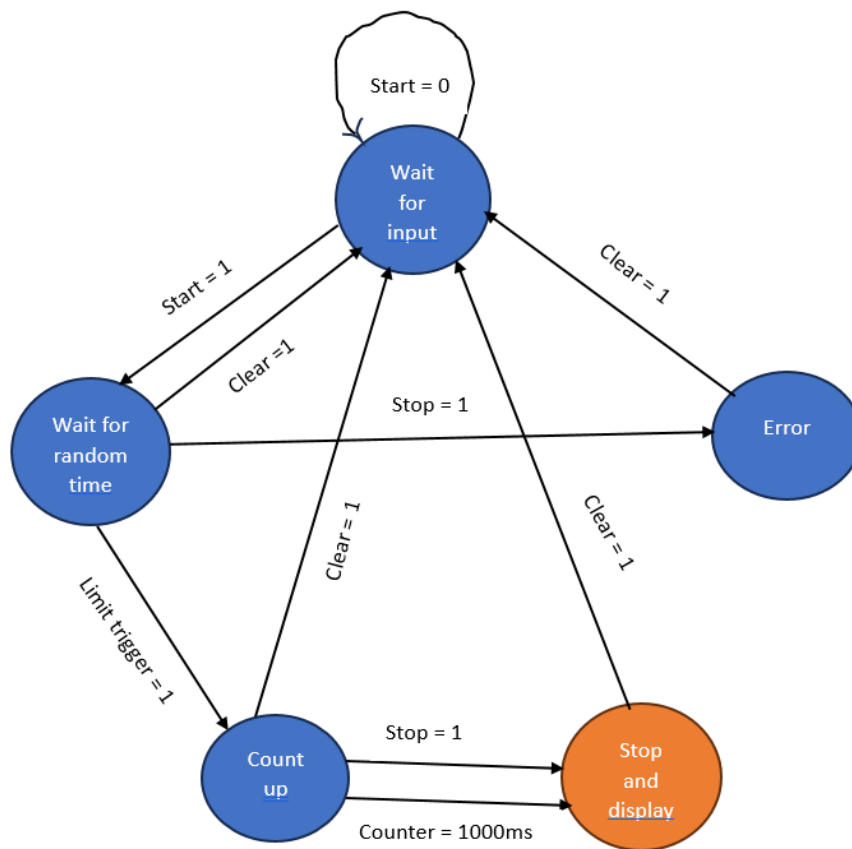


Fig: State diagram of the system

## 1.4 IMPLEMENTATION

The main module whose entity name is "human reaction," acts as the core connector, linking the ten sections and enabling seamless signal flow between modules. It achieves this by instantiating each module through entity declaration and port mapping to the algorithm. This integration ensures efficient communication and collaboration among all components, facilitating a cohesive and interconnected system design. The module involves measuring the time it takes for a person to respond to a stimulus (LED lights). It takes inputs from the start, stop, and clear buttons, along with a 100MHz system clock. It generates a random number using a random number generator, initializes the system, and triggers the stimulus display. The user needs to press the stop button when they see the stimulus, and the module measures and displays the reaction time on a seven-segment display. The

module utilizes debounce circuits to avoid noise in button inputs and divides the clock frequency to manage time intervals accurately. The system goes through various stages (INITIALIZE, RANDOM\_TIME, COUNTING, ERROR, and HOLD) to manage the stimulus display and reaction time measurement process effectively.

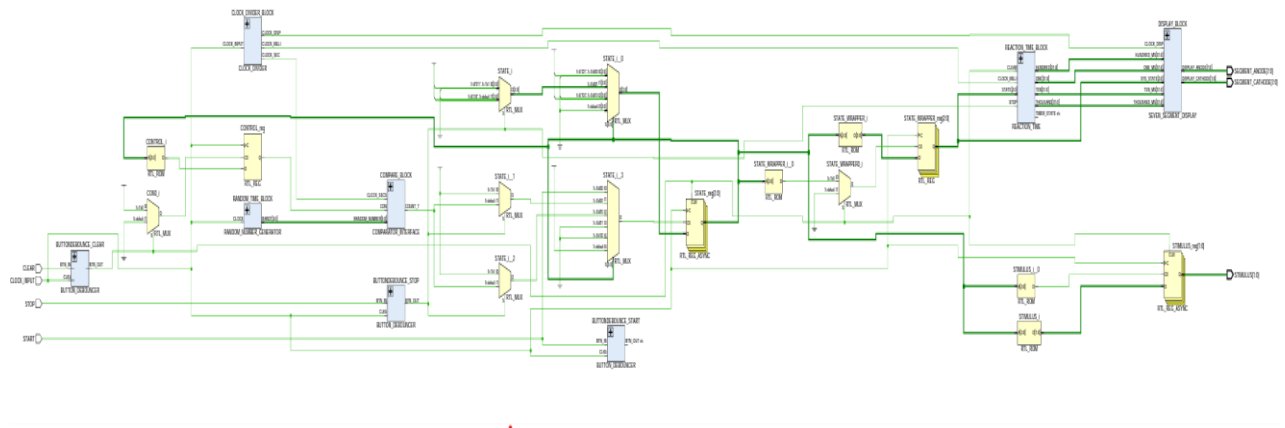


Fig: Schematic of human reaction timer

#### 1.4.1 CLOCK DIVIDER

The clock divider is responsible for generating three different clock signals from a single 100MHz input clock as the Nexys-4DDR board runs at 100MHz. The clock divider module functions by taking a 100MHz input clock signal (CLOCK\_INPUT) and generates three distinct output clock signals with lower frequencies: CLOCK\_MILLI, CLOCK\_DISP, and CLOCK\_SEC. The main principle behind the clock divider involves employing counters to divide the input clock frequency and toggle the output clock signals correspondingly. Each output clock signal operates with its own process independently to achieve the desired frequency. CLOCK\_MILLI (1KHz) is produced by a counter (COUNT\_50000) incrementing on the rising edge of CLOCK\_INPUT, toggling the output signal after reaching 50,000, resulting in a 1KHz clock signal with a 1-millisecond period. CLOCK\_DISP (2 KHz) uses a counter (COUNT\_25000) that reaches 25,000 before toggling the output, producing a 2 KHz clock signal with a 0.5-millisecond (500 microseconds) period. CLOCK\_SEC (1Hz) is generated by a counter (COUNT\_100000000) reaching 100,000,000, toggling the output, creating a 1Hz clock signal with a 1-second period. These processes run concurrently and independently, providing precise and synchronized timing for various purposes within the system, such as event timing, process control, and display refreshing, ensuring efficient operation and coordination of system components.

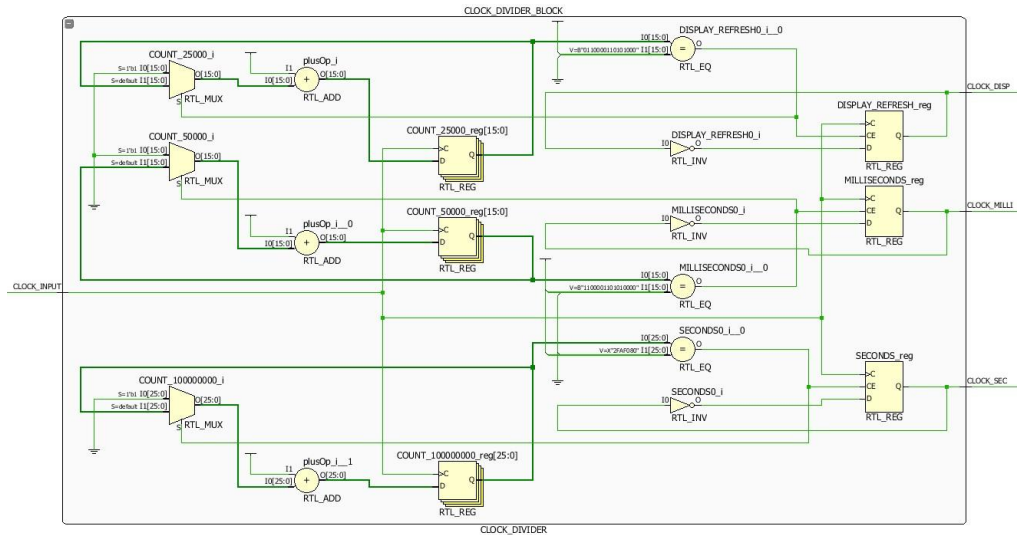


Fig: Clock divider block

### 1.4.2 SEVEN SEGMENT DISPLAY

The "SEVEN\_SEGMENT\_DISPLAY" entity represents a hardware module designed to control a seven-segment display of the Nexys-4DDR board. It operates based on a 2 KHz clock signal (CLOCK\_DISP), a 3-bit signal denoting the system state (STATES), and integer inputs (ONE\_MS, TEN\_MS, HUNDRED\_MS, and THOUSAND\_MS) representing millisecond values. The module's outputs, "DISPLAY\_ANODE" and "DISPLAY\_CATHODE," activate specific LEDs and segments, respectively, to display alphanumeric characters. In the initial state ("000"), it cycles through segments to display 'H' and 'I', showing a dash for undefined cases. Upon entering the "001" state, the display turns off, preparing for timer initiation. In "010" or "100" states, it functions as a countdown timer, showing the countdown with 1ms, 10ms, 100ms, or 1000ms resolution, depending on the corresponding input values. During the "011" state (Error State), all segments turn on rapidly. The module utilizes the rising edge of CLOCK\_DISP, and its "SEGMENT" variable facilitates cycling through segments during countdown states.

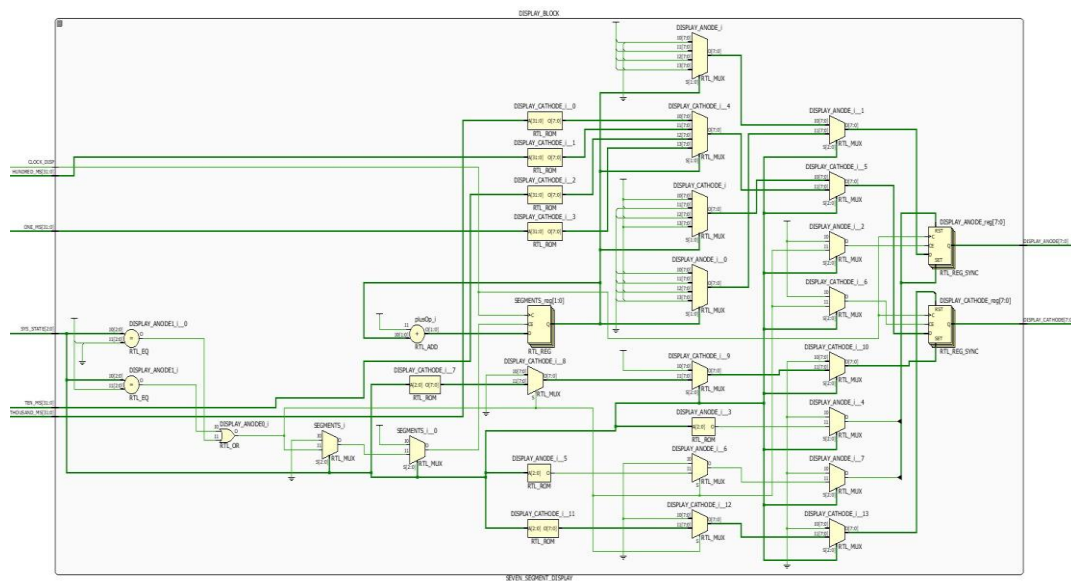


Fig: Seven segment display block



### 1.4.3 RANDOM GENERATOR BLOCK

The "RANDOM\_NUMBER\_GENERATOR" entity represents a Random Number Generator (RNG) implemented as a Linear Feedback Shift Register (LFSR). Operating on a 4-bit output "RAND1" and an input clock signal "CLOCK," the module generates a pseudo-random sequence of 4-bit numbers on each rising edge of the input clock. The LFSR utilizes two 4-bit registers, "LFSR\_REG1" and "LFSR\_REG2." During each clock cycle, the value of "LFSR\_REG2" is shifted into "LFSR\_REG1," introducing randomness into the generated sequence. The XOR operation between the least significant bits of "LFSR\_REG1" (bits 0 and 1) produces feedback, crucial for the LFSR's random behaviour. The feedback is then concatenated with the three most significant bits of "LFSR\_REG1" (bits 3 to 1), creating the next random value. The 4-bit "RAND1" output represents the generated random number, which continuously changes with each rising clock edge. The initial values of "LFSR\_REG1" and "LFSR\_REG2" ("1010") determine the sequence of random numbers produced by the generator, and the period of the LFSR is limited to 15 clock cycles ( $2^4 - 1$ ) for this 4-bit implementation.

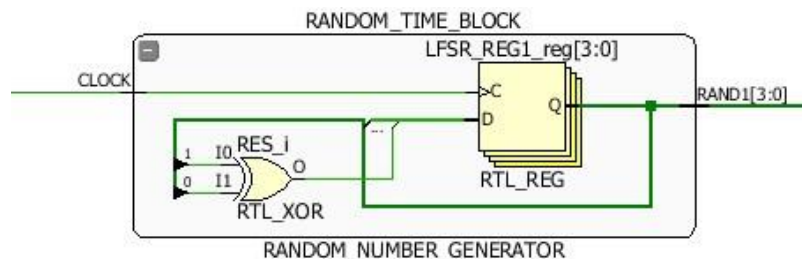


Fig: Random number generator block

### 1.4.4 LATCH

The entity "LATCH" defines a 4-bit latch module with one control input (C) and two data ports (RAND\_IN and RAND\_OUT). This latch stores and retains data when the control signal (C) is high, updating the output data only when the control is asserted. The LATCH entity has three ports: RAND\_IN, a 4-bit input representing the data to be latched; C, a single-bit control input; and RAND\_OUT, a 4-bit output for the latched data. The STRUCTURE architecture is used to describe the latch's functionality. Inside the process, the latch checks the value of the control signal (C). If C is high, the latching operation takes place, and the data from RAND\_IN is stored in RAND\_OUT. If C is low, the latching operation is skipped, and the output data remains unchanged. The block also explicitly sets RAND\_OUT to "0010" when RAND\_IN equals "0001" (binary value 1) on the rising edge of the control signal. For other input data values, the latch transfers the 4-bit data from RAND\_IN to RAND\_OUT while the control signal is asserted.

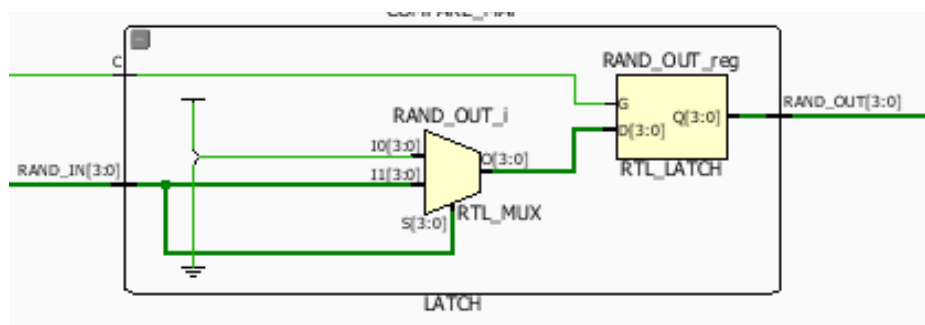


Fig: Latch block

#### 1.4.5 FOUR-BIT COMPARATOR

This entity describes 4-bit comparator module called "COMPARATOR\_4\_BIT," which compares two 4-bit inputs, NUM1 and NUM2, and generates an output signal, COMP, that indicates whether NUM1 is larger than or equal to NUM2. A clock signal (CLK2) and an enable signal (EN) are used to activate the comparator simultaneously. When EN is '1', the output COMP is set to '0', indicating that the comparator is turned off. When EN is set to '0', the comparison is done, and if NUM1 is larger than or equal to NUM2, COMP is set to '1'; otherwise, it is set to '0'. The outcome is available on CLK2's increasing edge.

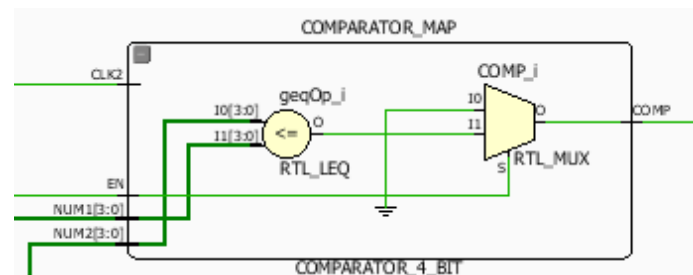


Fig: Four-bit comparator block

#### 1.4.6 FOUR-BIT COUNTER

The module named "FOUR\_BIT\_COUNTER" defines a four-bit counter that increments its value on the rising edge of a 1Hz clock signal (CLK\_1HZ) and can be reset to zero when the reset signal (RESET) is asserted. The module outputs the 4-bit counter value (C\_OUT) representing the current count. During the reset, the counter is set to zero, and on each rising edge of the 1Hz clock, it increments its value. The counter module functions as a simple 4-bit up counter with reset capability, providing a binary count starting from zero.

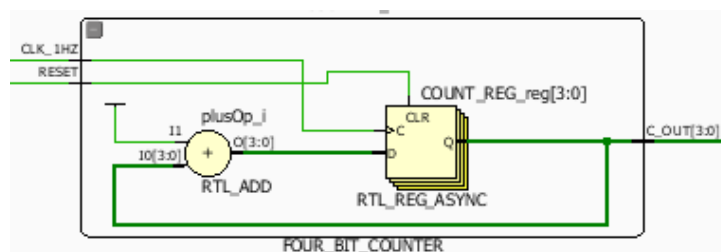


Fig: Four-bit counter block

#### 1.4.7 COUNTER-MAPPING

The entity named "COUNTER\_MAPPING," maps a 4-bit counter module called "FOUR\_BIT\_COUNTER" to external ports. The counter module has inputs for the clock (CLK) and reset (RESET) signals, and it provides a 4-bit output (C\_OUT) representing the counter's current value. The entity "COUNTER\_MAPPING" has two ports: CLK and RESET as input signals, and C\_OUT as the 4-bit output signal. Inside the architecture "BEHAVIORAL," the component "FOUR\_BIT\_COUNTER" is declared and instantiated using the "COUNTER\_MAP" instance. This instance maps the external signals CLK and RESET to the corresponding inputs of the counter module, and it also connects the C\_OUT output of the counter module to the C\_OUT output of the entity "COUNTER\_MAPPING." In summary, this VHDL code establishes a mapping between the 4-bit counter module "FOUR\_BIT\_COUNTER" and the entity "COUNTER\_MAPPING," allowing the counter module to be used and connected to external circuits through the entity's ports CLK, RESET, and C\_OUT.

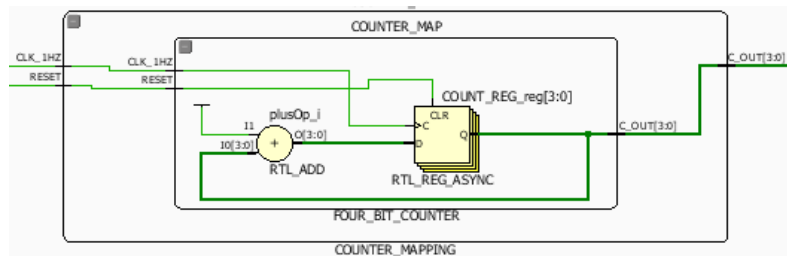


Fig: Counter mapping block

#### 1.4.8 COMPARATOR INTERFACE

The entity named "COMPARATOR\_INTERFACE" integrates several components to create a system for comparing a randomly generated 4-bit number with a 4-bit counter output. The components include a "RANDOM\_NUMBER\_GENERATOR" for generating the random number, a "COUNTER\_BLOCK" for the counter output, a "LATCH" for storing the random number, and a "COMPARATOR\_4\_BIT" for performing the comparison. The system uses a common clock signal (CLOCK\_SEC) and control signal (CON) to synchronize the components. The random number is latched when the control signal is asserted, and then the comparator compares the latched random number with the counter output. The result of the comparison is provided as the "COUNT\_T" output.

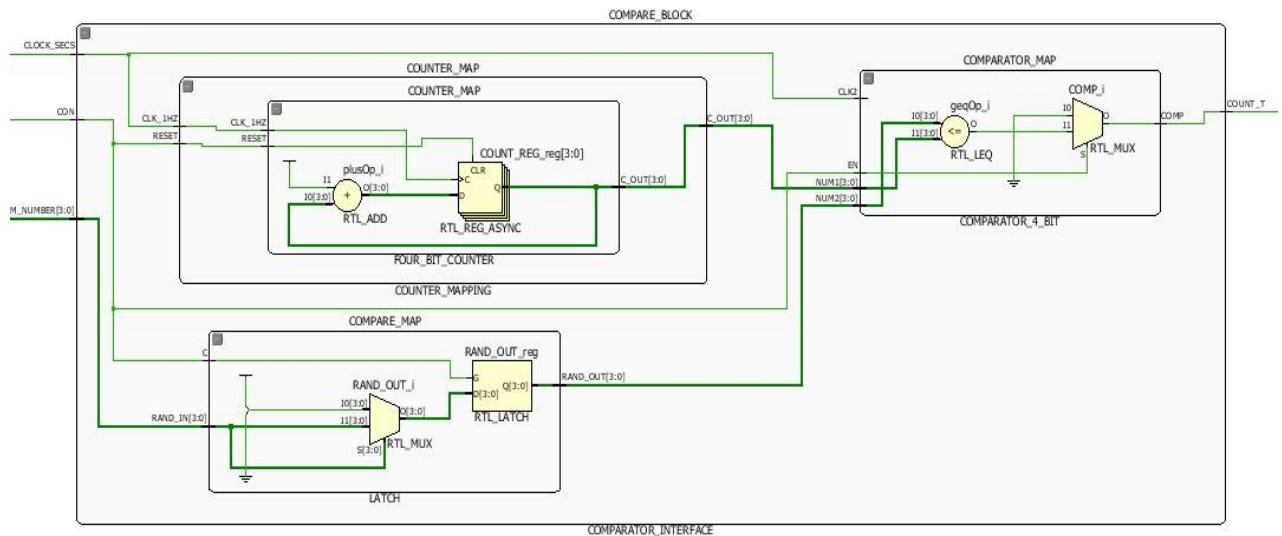


Fig: Compare interface block

#### 1.4.9 BUTTON DEBOUNCE

The entity "BUTTON\_DEBOUNCER" describes a button debouncer module, which is used to remove button bounce in digital systems. It accepts a clock signal (CLK5) and a button signal (BTN\_IN) as inputs and outputs a debounced button signal (BTN\_OUT). A counter is used to identify steady button states over a priest number of cycles, blocking out fast variations produced by button bouncing. BTN\_OUT, the debounced output, depicts the button's steady state, ensuring clean and dependable input for digital systems.

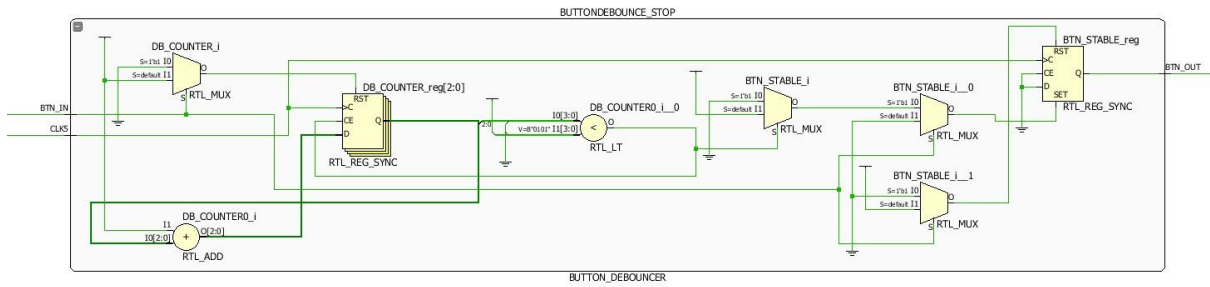


Fig: Button debouncer block

#### 1.4.10 REACTION TIME

The module called "REACTION\_TIME" implements a reaction time measurement. It utilizes a 1 kHz clock input and stop and clear button signals to measure the elapsed time in milliseconds. The module increments counters for milliseconds, tens, hundreds, and thousands until the stop button is pressed, indicating a hold state. The counters reset and update accordingly to measure the reaction time, and the measured time in milliseconds is provided as outputs. When the timer reaches 1000 milliseconds (1 second), the module goes into a hold state, indicated by the TIMER\_STATE output. The module effectively measures and displays the reaction time from the moment the stop button is pressed until 1000 milliseconds are reached.

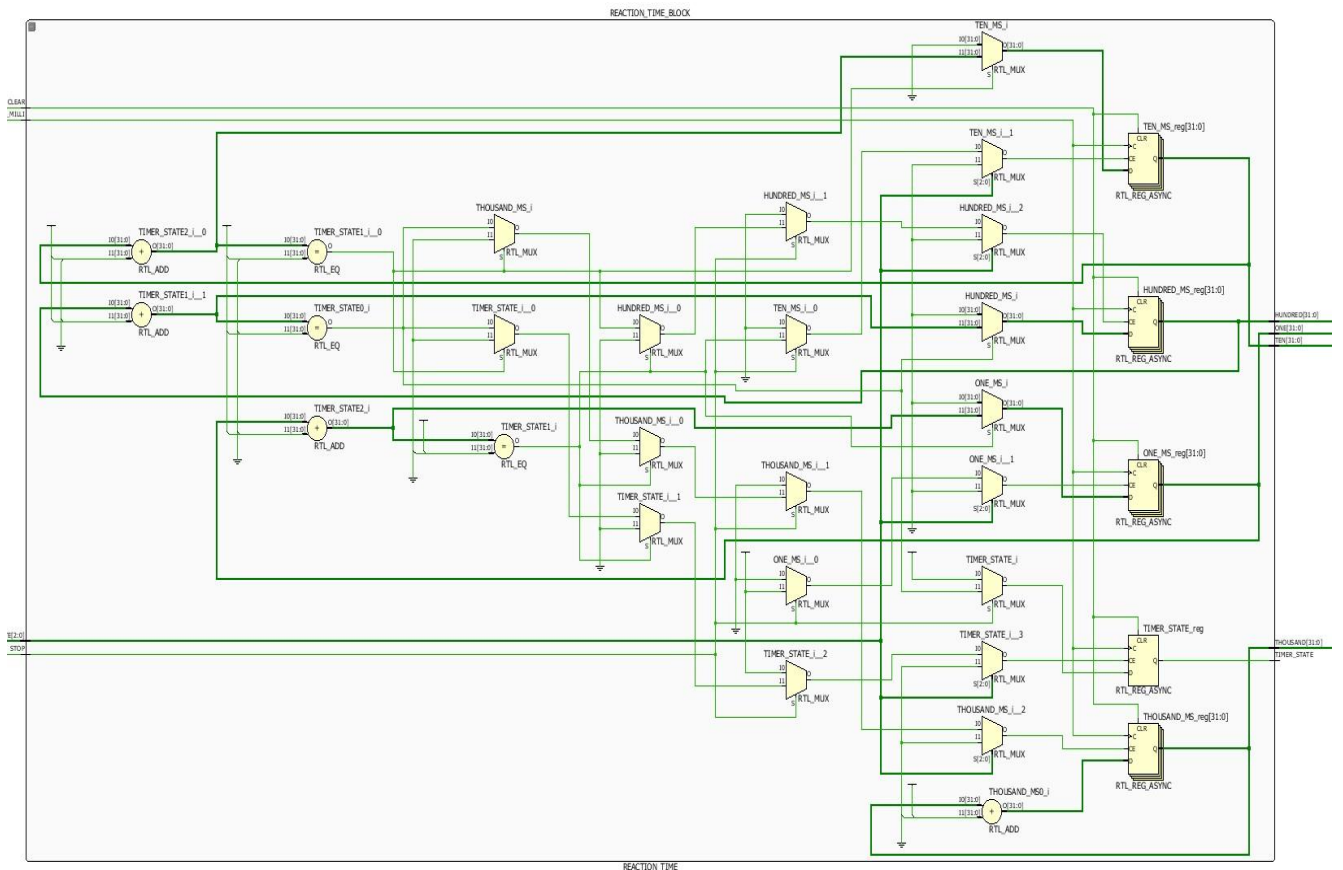


Fig: Reaction time block

#### 1.4.11 POWER CONSUMPTION AND RESOURCE UTILIZATION

Power estimation is a crucial step in electronic design, determining the power consumption of a circuit before it is physically implemented on hardware. A synthesized netlist is generated from a higher-level description like RTL code using a synthesis tool, calculating the power consumed by different components of the chip based on their activity, switching, and leakage characteristics. The table presented in the picture is divided into two sections: On-Chip Power and Junction Temperature. The On-Chip Power section displays the percentage of power consumed by different components, such as dynamic power, static power, signals power, and logic power. This breakdown helps identify power-hungry parts of the design, enabling designers to optimize power usage and potentially reduce overall power consumption.

Junction Temperature is a critical parameter that affects the reliability and performance of the chip, calculated based on power dissipation and thermal characteristics. Knowing the junction temperature helps designers ensure the chip operates within safe temperature limits. Academic analytics provide valuable insights into power estimation results, such as a total on-chip power consumption of 1.028 W, which is considered relatively low for power-efficient designs. High dynamic power (49%) indicates significant switching activity, suggesting the need for power-saving techniques like clock gating, voltage scaling, or power gating to reduce dynamic power. Low static power consumption (8%) indicates low leakage current and may not require additional power-saving techniques like body biasing or subthreshold logic for static power reduction.

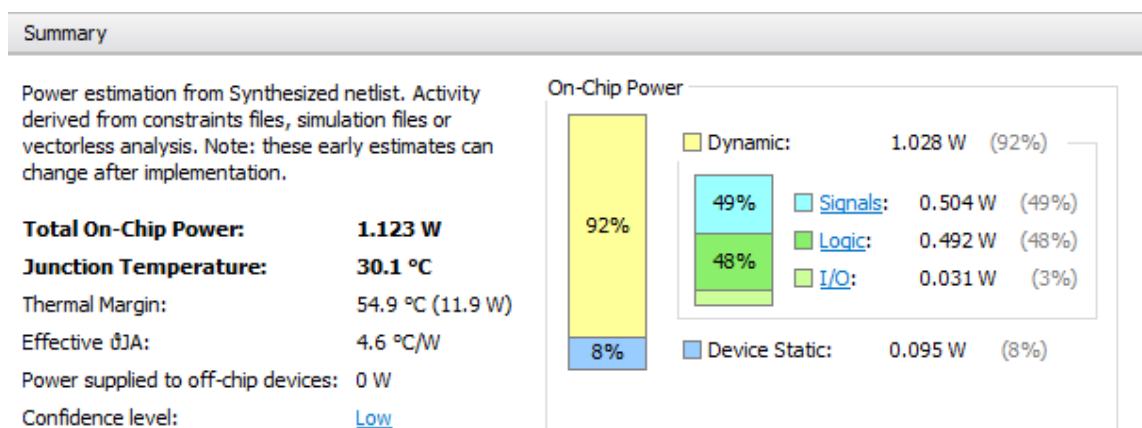


Fig: Power consumption

Utilization data is used to assess the efficiency and effectiveness of a resource in its operation. The table presented in the picture contains utilization data for different resources, displaying metrics such as utilization rate, score, grade, cost, weight, impact, and ranking. Utilization data helps organizations identify areas of improvement, optimization, and innovation, enabling informed decisions to increase efficiency and productivity. Comparing utilization data across different resources or time periods enables benchmarking and setting targets for improvement. However, obtaining accurate and consistent utilization data can be challenging. Data collection and analysis must be reliable and valid, with standardizing data definitions and measurement methods ensuring consistency. Additionally, considering relevant context and criteria is essential for meaningful comparisons.

Utilization Details - Hierarchical							
Utilization	Name	Signals (W)	Data (W)	Clock Enable (W)	Set/Reset (W)	Logic (W)	I/O (W)
1.028 W (92% of total)	HUMAN_REACTION						
0.539 W (48% of total)	CLOCK_DIVIDER_BLOCK (CLOCK_DIVIDER)	0.221	0.221	<0.0001	<0.0001	0.318	<0.0001
0.118 W (11% of total)	Leaf Cells (38)						
0.081 W (7% of total)	COMPARE_BLOCK (COMPARATOR_INTERFACE)	0.039	0.039	<0.0001	<0.0001	0.042	<0.0001
0.075 W (7% of total)	DISPLAY_BLOCK (SEVEN_SEGMENT_DISPLAY)	0.045	0.042	0.002	0.001	0.03	<0.0001
0.07 W (6% of total)	RANDOM_TIME_BLOCK (RANDOM_NUMBER_GENERATOR)	0.058	0.058	<0.0001	<0.0001	0.012	<0.0001
0.053 W (5% of total)	BUTTONDEBOUNCE_CLEAR (BUTTON_DEBOUNCER)	0.03	0.023	<0.0001	0.007	0.023	<0.0001
0.048 W (4% of total)	BUTTONDEBOUNCE_STOP (BUTTON_DEBOUNCER_0)	0.024	0.023	0.001	<0.0001	0.024	<0.0001
0.044 W (4% of total)	REACTION_TIME_BLOCK (REACTION_TIME)	0.008	0.008	<0.0001	<0.0001	0.036	<0.0001

Fig: Utilization of resource

## 1.5 SIMULATION

To simulate the output of the hardware, test benches of each module must be carried out. A test bench is a VHDL-based simulation environment used to validate the operation of any digital circuit or design. It stimulates the design and compares its outputs to predicted results, letting designers to test and debug the concept before it is implemented in hardware. The test bench assures functional accuracy, detects mistakes, allows for early testing, and aids with performance analysis. It is a vital tool for testing complicated designs efficiently and thoroughly before hardware implementation, assuring reliability and accuracy.

### 1.5.1 CLOCK DIVIDER TEST BENCH

The test bench clock divider aims to validate the CLOCK\_DIVIDER module, which generates three clock signals with different frequencies based on a 10 ns input clock. The test bench, named CLOCK\_DIVIDER\_TB, ensures simulation and verification of the clock divider's behaviour under diverse conditions. The three output clocks, CLOCK\_DISP\_TB, CLOCK\_MILLI\_TB, and CLOCK\_SEC\_TB, are generated at specific frequencies relative to the input clock. CLOCK\_DISP\_TB matches the input clock, CLOCK\_MILLI\_TB is half its frequency (20 ns period), and CLOCK\_SEC\_TB is one-tenth its frequency (100 ns period). The test bench verifies the proper functioning of these output clocks according to the CLOCK\_DIVIDER module's specifications.

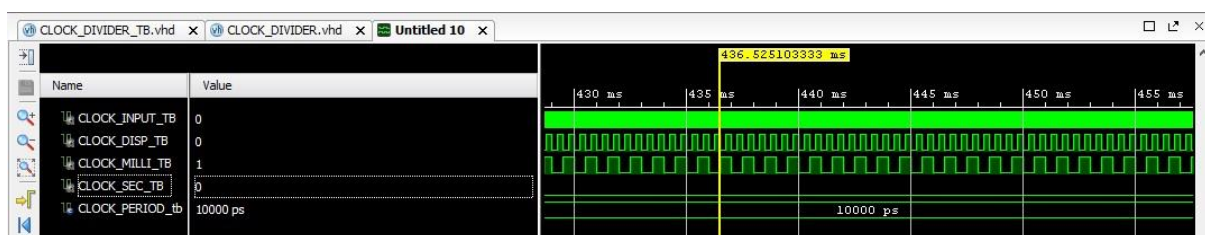


Fig: Testbench of clock divider

### 1.5.2 RANDOM NUMBER GENERATOR TEST BENCH

The test bench of random number generator is designed to validate the functionality of the "RANDOM\_NUMBER\_GENERATOR" component. It provides a clock signal (CLOCK) and monitors the output signal (RAND1). The test bench simulates clock cycles by alternating the CLOCK signal between '0' and '1' with a 10 ns period. The DUT port map connects the test bench signals to the RANDOM\_NUMBER\_GENERATOR component, allowing the test bench to drive the input clock and capture the random 4-bit values generated on the output RAND1. The simulation enables designers to verify the component's correct operation by analysing the sequence of random numbers produced.



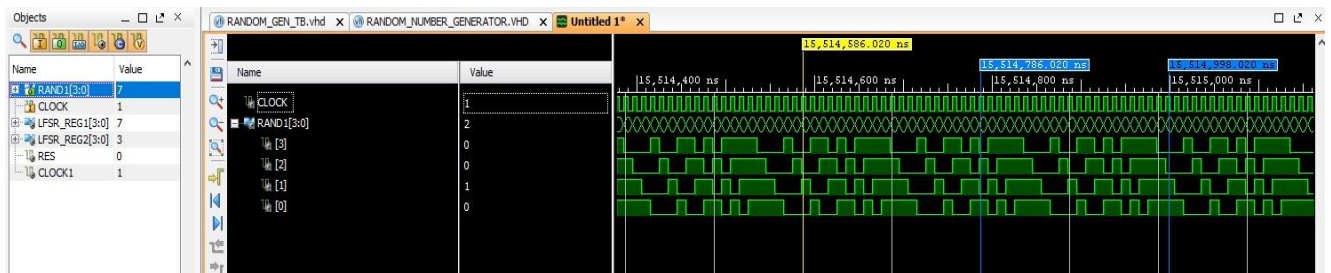


Fig: Testbench of random number generator

### 1.5.3 LATCH TEST BENCH

This test bench is done to verify the functionality of a 4-bit latch module (LATCH). It simulates different test cases, applying control (C\_tb) and input data (RAND\_IN\_tb) to the latch and observing the resulting 4-bit output (RAND\_OUT\_tb). The test bench generates a clock signal (CLOCK) with a 10 ns period and runs the simulation for 1000 ns. It initializes the input signals, waits for a few clock cycles, and then applies the test cases to the latch. The latch component processes the inputs during simulation, updating the output accordingly. The test bench assesses whether the latch behaves as expected, producing the correct output values based on the input data and control signals. The output signal (RAND\_OUT\_tb) captures the results of the latch operation during each test case.

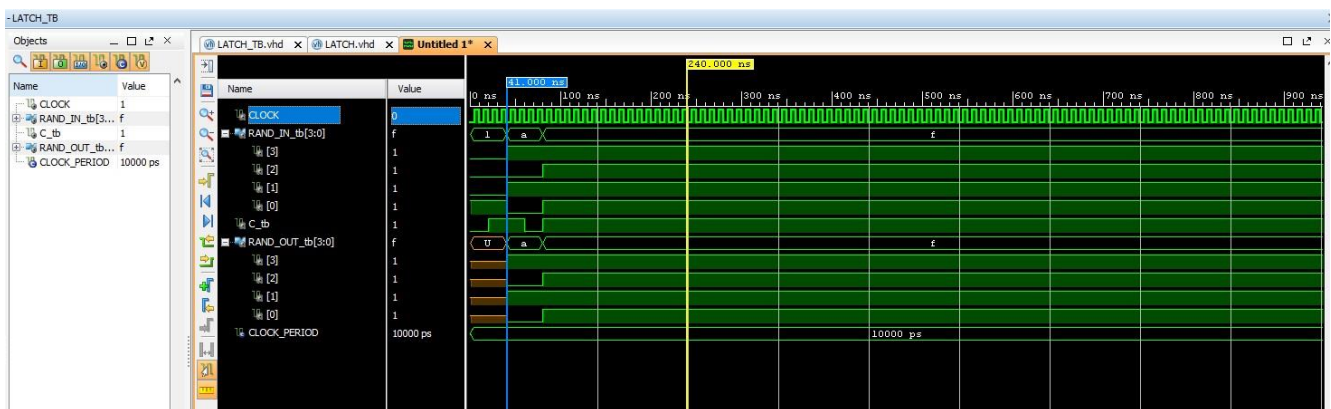


Fig: Testbench of latch

### 1.5.4 COUNTER TEST BENCH

This is a testbench designed to verify the functionality of a 4-bit counter module (FOUR\_BIT\_COUNTER). The test bench contains two processes: CLOCK\_GEN and COUNTER\_PROCESS. The CLOCK\_GEN process generates a 1 microsecond (1 us) clock signal (CLK\_1HZ\_TB) with alternating '0' and '1' values, running the simulation for 1000 us. The COUNTER\_PROCESS simulates the behaviour of the counter, initializing RESET\_TB to '1' for a reset, then releasing the reset after a delay. It waits for 500 us to test the counter output (C\_OUT\_TB) for 500 us. The FOUR\_BIT\_COUNTER component increments its output on each rising edge of the input clock (CLK\_1HZ\_TB) and can be reset to zero by setting RESET\_TB to '1'. The test bench verifies the counter's behaviour by providing input signals (RESET\_TB and CLK\_1HZ\_TB) and monitoring the output signal (C\_OUT\_TB) to ensure correct counting and the expected output sequence.

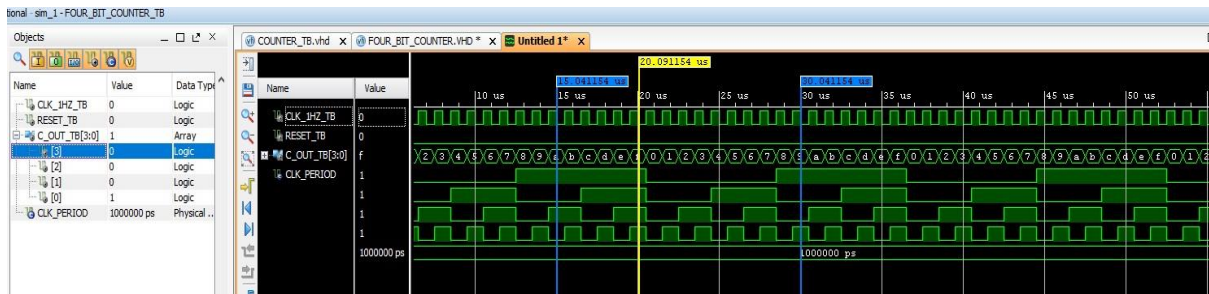


Fig: Testbench of counter

### 1.5.5 COMPARATOT TEST BENCH

Here a testbench designed to verify the functionality of a 4-bit comparator module (COMPARATOR\_4\_BIT). The test bench includes two processes: CLOCK\_PROCESS, generating a 100 MHz clock signal (CLK2\_TB), and COMPARATOR\_PROCESS, which tests various cases by providing different 4-bit numbers (NUM1\_TB and NUM2\_TB) and an enable signal (EN\_TB). The comparator checks for equality or magnitude between NUM1\_TB and NUM2\_TB and outputs the result as a single-bit on COMP\_TB. The test cases involve equal, greater, and less than comparisons, and the simulation observes the changes in COMP\_TB accordingly. The output of this test bench, COMP\_TB, allows designers to verify the proper functioning of the 4-bit comparator module (COMPARATOR\_4\_BIT) in performing accurate comparisons between 4-bit numbers.

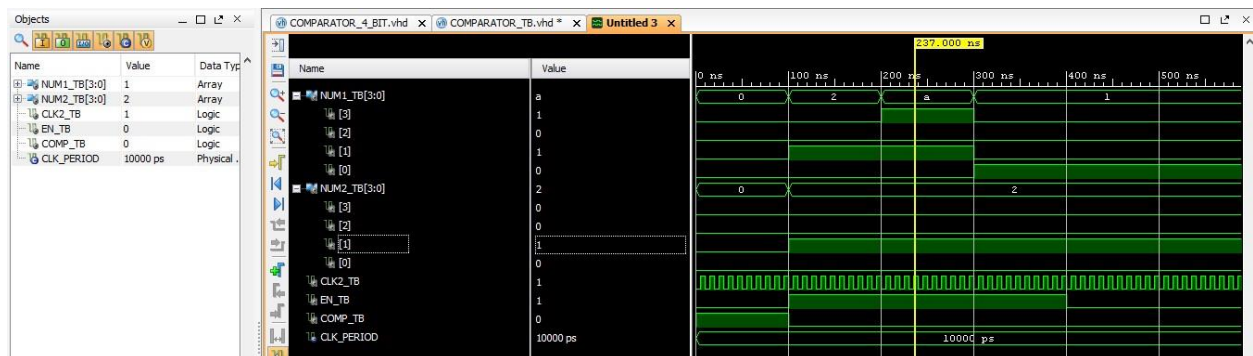


Fig: Testbench of comparator

### 1.5.6 REACTION TIME TEST BENCH

The figure below is a test bench designed to verify the functionality of a reaction time measurement module (REACTION\_TIME). The test bench includes two processes: CLK\_PROCESS, generating a 1 MHz clock signal (CLOCK\_MILLI\_TB), and REACTION\_TIME\_PROCESS, which tests different cases by providing control signals (STOP\_TB, CLEAR\_TB) and state information (STATE\_TB) to the reaction time module. The module measures time in milliseconds (ONE\_TB, TEN\_TB, HUNDRED\_TB, and THOUSAND\_TB) and updates its state (TIMER\_STATE\_TB) based on the test cases applied during the simulation. The test bench verifies the behaviour of the reaction time measurement module (REACTION\_TIME) by providing input signals (CLOCK\_MILLI\_TB, STOP\_TB, CLEAR\_TB, and STATE\_TB) and monitoring the output signals (ONE\_TB, TEN\_TB, HUNDRED\_TB, THOUSAND\_TB, and TIMER\_STATE\_TB) to ensure that the time measurement and state transitions occur as expected for different test cases. The simulation includes three test cases which are test case 1 is where the state is



set to "010" (Hold State), and the timer counts for 5000 milliseconds (5 seconds) without being stopped. After the specified duration, the ONE\_TB, TEN\_TB, HUNDRED\_TB, and THOUSAND\_TB signals should show the appropriate values based on the elapsed time. Test case 2 where the state remains "010" (Hold State), and the timer counts for 15000 milliseconds (15 seconds). However, this time, the STOP\_TB signal is set to '1', stopping the timer before the 15-second duration is complete. The output signals (ONE\_TB, TEN\_TB, HUNDRED\_TB, and THOUSAND\_TB) should reflect the time measurement up to the point of stopping. Test Case 3: The state is set to "001" (Some other state). In this case, the timer should not count, and the output signals (ONE\_TB, TEN\_TB, HUNDRED\_TB, and THOUSAND\_TB) should not change. Overall, the output signals from this test bench (ONE\_TB, TEN\_TB, HUNDRED\_TB, THOUSAND\_TB, and TIMER\_STATE\_TB) allow designers to verify whether the reaction time measurement module (REACTION\_TIME) functions correctly in measuring time accurately and updating its state as expected based on the test cases applied during the simulation.

Fig: Testbench of reaction time

The output of the test bench (SEVEN\_SEGMENT\_DISPLAY\_TB) consists of two signals: DISPLAY\_ANODE\_TB and DISPLAY\_CATHODE\_TB, which control the segments of a seven-segment display. The specific patterns displayed on the seven-segment display depend on the input time values and the module's state. In the first test case, it may show initialization patterns. In the second test case represents a "Random Wait Interlock" state with specific patterns. The third and fourth test cases display time values in milliseconds based on input signals ONE\_MS\_TB, TEN\_MS\_TB, HUNDRED\_MS\_TB, and THOUSAND\_MS\_TB. The fifth test case likely shows an error indication, such as turning off all segments or displaying an error pattern. The output patterns serve to verify whether the SEVEN\_SEGMENT\_DISPLAY module correctly interprets input time values, displays them accurately, and responds appropriately to system states, aiding designers in identifying and addressing potential implementation issues.

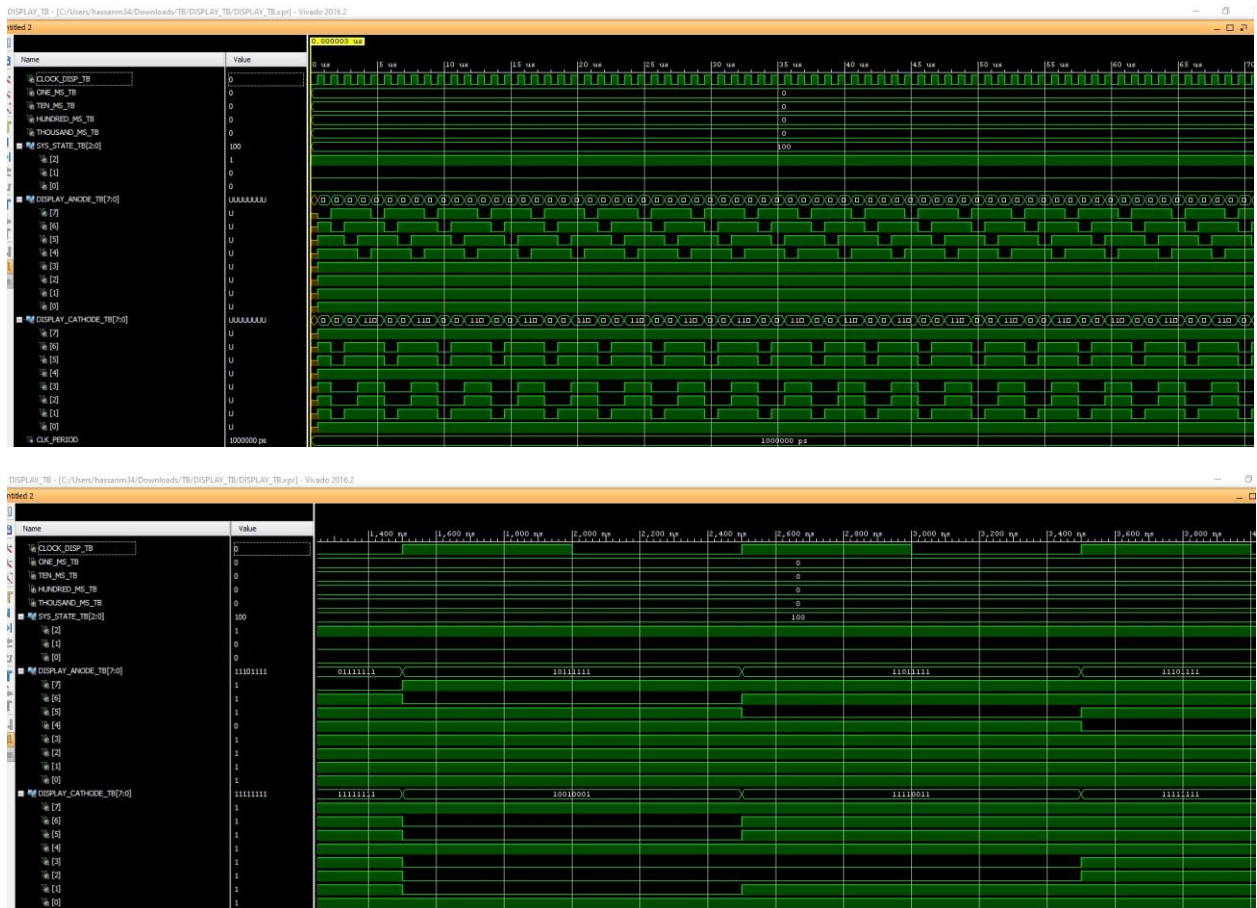


Fig: Testbench of seven segment display

### 1.5.7 BUTTON DEBOUNCE TEST BENCH

This testbench simulates the behaviour of the BUTTON\_DEBOUNCER entity by providing different input values (BTN\_IN) simulating button presses by the user and observing the output behaviour (BTN\_OUT). The simulation includes test cases where the button is pressed and released multiple times to test the debouncing functionality. After the pre-set debounce cycles, BTN\_OUT\_TB becomes '1', indicating that the BUTTON\_DEBOUNCER has detected a stable button press.

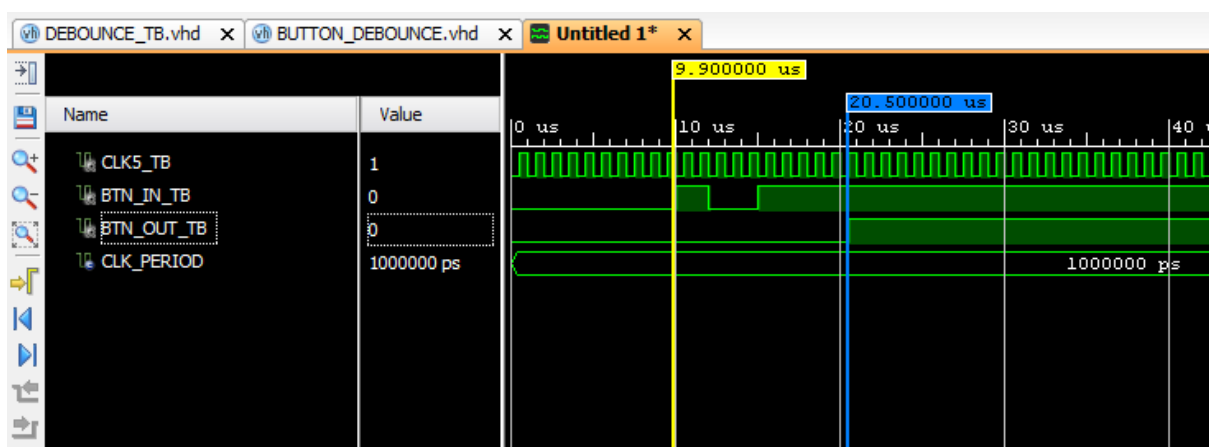
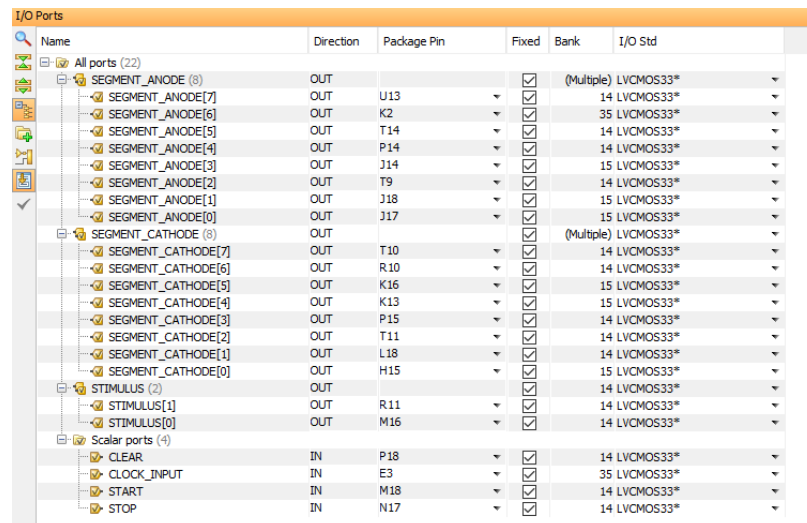


Fig: Testbench of Button Debounce

## 1.6 HARDWARE IMPLEMENTATION

The project was successfully programmed on the NEXYS 4 DDR board with all the states and outputs functioning as intended. Hardware implementation demonstrated in the uploaded [group demonstration video](#).



Name	Direction	Package Pin	Fixed	Bank	I/O Std
<b>All ports (22)</b>					
SEGMENT_ANODE (8)	OUT		<input checked="" type="checkbox"/>	(Multiple)	LVC MOS33*
SEGMENT_ANODE[7]	OUT	U13	<input checked="" type="checkbox"/>		14 LVC MOS33*
SEGMENT_ANODE[6]	OUT	K2	<input checked="" type="checkbox"/>		35 LVC MOS33*
SEGMENT_ANODE[5]	OUT	T14	<input checked="" type="checkbox"/>		14 LVC MOS33*
SEGMENT_ANODE[4]	OUT	P14	<input checked="" type="checkbox"/>		14 LVC MOS33*
SEGMENT_ANODE[3]	OUT	J14	<input checked="" type="checkbox"/>		15 LVC MOS33*
SEGMENT_ANODE[2]	OUT	T9	<input checked="" type="checkbox"/>		14 LVC MOS33*
SEGMENT_ANODE[1]	OUT	J18	<input checked="" type="checkbox"/>		15 LVC MOS33*
SEGMENT_ANODE[0]	OUT	J17	<input checked="" type="checkbox"/>		15 LVC MOS33*
SEGMENT_CATHODE (8)	OUT		<input checked="" type="checkbox"/>	(Multiple)	LVC MOS33*
SEGMENT_CATHODE[7]	OUT	T10	<input checked="" type="checkbox"/>		14 LVC MOS33*
SEGMENT_CATHODE[6]	OUT	R10	<input checked="" type="checkbox"/>		14 LVC MOS33*
SEGMENT_CATHODE[5]	OUT	K16	<input checked="" type="checkbox"/>		15 LVC MOS33*
SEGMENT_CATHODE[4]	OUT	K13	<input checked="" type="checkbox"/>		15 LVC MOS33*
SEGMENT_CATHODE[3]	OUT	P15	<input checked="" type="checkbox"/>		14 LVC MOS33*
SEGMENT_CATHODE[2]	OUT	T11	<input checked="" type="checkbox"/>		14 LVC MOS33*
SEGMENT_CATHODE[1]	OUT	L18	<input checked="" type="checkbox"/>		14 LVC MOS33*
SEGMENT_CATHODE[0]	OUT	H15	<input checked="" type="checkbox"/>		15 LVC MOS33*
STIMULUS (2)	OUT		<input checked="" type="checkbox"/>		14 LVC MOS33*
STIMULUS[1]	OUT	R11	<input checked="" type="checkbox"/>		14 LVC MOS33*
STIMULUS[0]	OUT	M16	<input checked="" type="checkbox"/>		14 LVC MOS33*
<b>Scalar ports (4)</b>					
CLEAR	IN	P18	<input checked="" type="checkbox"/>		14 LVC MOS33*
CLOCK_INPUT	IN	E3	<input checked="" type="checkbox"/>		35 LVC MOS33*
START	IN	M18	<input checked="" type="checkbox"/>		14 LVC MOS33*
STOP	IN	N17	<input checked="" type="checkbox"/>		14 LVC MOS33*

Fig: Hardware I/O Planning.

## 1.7 CONCLUSION

In conclusion, the report present successful FPGS development and implementation of measure human response timer after exposure to a visual stimulus on a Nexys-4DDR board. The design was structured into ten distinct sections, each focusing on a specific aspect of the circuit, which were independently developed and later integrated into the final system.

The FPGA implementation featured three input pushbuttons, stimulus LEDs, and a seven-segment LED display to present relevant information. The system flowchart and state diagram provide a clear representation of the circuit's behaviour and transitions, ensuring a comprehensive understanding of the human reaction timer's operation.

The report also discussed the modelling of the system using state diagrams, highlighting the importance of system modelling in risk minimization and requirement analysis.

The implementation section described various modules, including the clock divider, seven-segment display, random number generator, latch, counter, comparator, debounce circuit, and the reaction time measurement module. Each module's functionality was verified through detailed test benches, ensuring accurate and reliable operation prior to programming the device.

The power consumption and resource utilization analysis indicated that the design was relatively power-efficient, with low static power and moderate dynamic power. The utilization data provided insights into the efficiency of resource usage and potential areas for optimization and improvement.

Overall, the report presented a systematic and organized approach to designing, implementing, and testing the human reaction timer circuit. The circuit serves as a valuable tool for studying cognitive responses and can find applications in various fields where reaction time measurements are essential."

## SECTION II

### APPENDIX

#### 2.1.1 CLOCK DIVIDER

```
-----  
-- Project Name: FPGA Coursework  
-- Group: Flip Flops  
-- Create Date: 25.07.2023 14:30:12  
-- Module Name: DISPLAY  
-- Target Devices: Nexys-4-DDR  
-- Additional Comments:  
-----  
  
library IEEE;  
use IEEE.STD_LOGIC_1164.ALL;  
use IEEE.NUMERIC_STD.ALL;  
  
entity CLOCK_DIVIDER is  
    port (  
        CLOCK_INPUT: in std_logic; -- 100MHz Input Clock Signal  
        CLOCK_DISP: out std_logic; -- 2KHz Clock output for refreshing display every 0.5ms  
        CLOCK_MILLI: out std_logic; -- 1KHz Clock output for Counting in milli seconds  
        CLOCK_SEC: out std_logic; -- 1Hz Clock output for counting in seconds  
    );  
end CLOCK_DIVIDER;  
  
architecture BEHAVIORAL of CLOCK_DIVIDER is  
    signal SECONDS: std_logic := '0';  
    signal MILLISECONDS: std_logic := '0';  
    signal DISPLAY_REFRESH: std_logic := '0';  
begin  
  
        CLOCK_MILLI <= MILLISECONDS;  
        CLOCK_DISP <= DISPLAY_REFRESH;  
        CLOCK_SEC <= SECONDS;  
  
        CLOCK_1Hz: process(CLOCK_INPUT)
```

```

variable COUNT_100000000: unsigned (25 downto 0) := "000000000000000000000000"; -- Count to 100,000,000
for 1KHz Clock

```

```

begin
    if ( rising_edge (CLOCK_INPUT)) then
        if COUNT_100000000 = "10111110101111000010000000" then--counting to 100,000,000
            SECONDS <= not SECONDS;
            COUNT_100000000 := "000000000000000000000000";
        end if;
        COUNT_100000000 := COUNT_100000000 + 1;
    end if;
end process;

```

```

CLOCK_1KHz: process(CLOCK_INPUT)

```

```

variable COUNT_50000: unsigned (15 downto 0) := "0000000000000000"; -- Count to 50,000 for 1KHz Clock
begin
    if ( rising_edge (CLOCK_INPUT)) then
        if COUNT_50000 = "1100001101010000" then--counting to 50000
            MILLISECONDS <= not MILLISECONDS;
            COUNT_50000 := "0000000000000000";
        end if;
        COUNT_50000 := COUNT_50000 + 1;
    end if;
end process;

```

```

CLOCK_2KHz: process(CLOCK_INPUT)

```

```

variable COUNT_25000: unsigned (15 downto 0) := "0000000000000000"; -- Count to 25,000 for 2KHz Clock to refresh
displat every 0.5ms

```

```

begin
    if ( rising_edge (CLOCK_INPUT)) then
        if COUNT_25000 = "110000110101000" then--counting to 25,000
            DISPLAY_REFRESH <= not DISPLAY_REFRESH;
            COUNT_25000 := "0000000000000000";
        end if;
        COUNT_25000 := COUNT_25000 + 1;
    end if;
end process;

```

```

end BEHAVIORAL;

```

### 2.1.2 SEVEN SEGMENT DISPLAY

```
-- Project Name: FPGA Coursework
-- Group: Flip Flops
-- Create Date: 25.07.2023 14:30:12
-- Module Name: DISPLAY
-- Target Devices: Nexys-4-DDR
-- Additional Comments:
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity SEVEN_SEGMENT_DISPLAY is
    Port (CLOCK_DISP : in std_logic;
          ONE_MS : in integer;
          TEN_MS : in integer;
          HUNDRED_MS : in integer;
          THOUSAND_MS : in integer;
          SYS_STATE : in std_logic_vector(2 downto 0);
          DISPLAY_ANODE : out std_logic_vector(7 downto 0);
          DISPLAY_CATHODE : out std_logic_vector(7 downto 0)
    );
end SEVEN_SEGMENT_DISPLAY;

architecture BEHAVOIRAL of SEVEN_SEGMENT_DISPLAY is

begin
    process(CLOCK_DISP)
        variable SEGMENTS: unsigned(1 downto 0) := "00";
    begin
        if rising_edge(CLOCK_DISP) then
            if SYS_STATE = "100" then--INITIALIZATION STATE DISPLAY
                case (SEGMENTS) is
                    when "00" =>
                        DISPLAY_ANODE <= "01111111";
```

```

        DISPLAY_CATHODE <= "11111111";--BLANK

when "01" =>

        DISPLAY_ANODE <= "10111111";--TURN ON DISPLAY FOR "H"

        DISPLAY_CATHODE <= "10010001";

when "10" =>

        DISPLAY_ANODE <= "11011111";--TURN ON DISPLAY FOR "I"

        DISPLAY_CATHODE <= "11110011";

when "11" =>

        DISPLAY_ANODE <= "11101111";

        DISPLAY_CATHODE <= "11111111";--BLANK

when others =>

        DISPLAY_ANODE <= "11110000";

        DISPLAY_CATHODE <= "11111110";

end case;

SEGMENTS := SEGMENTS + 1;

elsif SYS_STATE = "101" then--RANDOM WAIT INTERCAL WAIT

        DISPLAY_ANODE <= "00001111";

        DISPLAY_CATHODE <= "11111111"; --ALL SEGMENTS OFF

elsif ((SYS_STATE = "111") or (SYS_STATE = "011")) then-- COUNTING OR HOLD States

case (SEGMENTS) is

        when "00" =>

                DISPLAY_ANODE <= "01111111";

case (THOUSAND_MS) is-- DISPLAY 1 second

        when 0 =>

                DISPLAY_CATHODE <= "00000011";

        when 1 =>

                DISPLAY_CATHODE <= "10011111";

        when others =>

                DISPLAY_CATHODE <= "11111110";

        end case;

when "01" =>

        DISPLAY_ANODE <= "10111111";--DISPLAY HUNDREDS Milliseconds

case (HUNDRED_MS) is

        when 0 =>

                DISPLAY_CATHODE <= "00000011"; --Display 0

        when 1 =>

```

```

        DISPLAY_CATHODE <= "10011111"; --Display 1
when 2 =>
        DISPLAY_CATHODE <= "00100101"; --Display 2
when 3 =>
        DISPLAY_CATHODE <= "00001101"; --Display 3
when 4 =>
        DISPLAY_CATHODE <= "10011001"; --Display 4
when 5 =>
        DISPLAY_CATHODE <= "01001001"; --Display 5
when 6 =>
        DISPLAY_CATHODE <= "11000001"; --Display 6
when 7 =>
        DISPLAY_CATHODE <= "00011111";--Display 7
when 8 =>
        DISPLAY_CATHODE <= "00000001";--Display 8
when 9 =>
        DISPLAY_CATHODE <= "00011001";--Display 9
when others =>
        DISPLAY_CATHODE <= "11111110";
end case;
when "10" =>
        DISPLAY_ANODE <= "11111101";--DISPLAY TENs Milliseconds
        case (TEN_MS) is
            when 0 =>
                DISPLAY_CATHODE <= "00000011";
            when 1 =>
                DISPLAY_CATHODE <= "10011111";
            when 2 =>
                DISPLAY_CATHODE <= "00100101";
            when 3 =>
                DISPLAY_CATHODE <= "00001101";
            when 4 =>
                DISPLAY_CATHODE <= "10011001";
            when 5 =>
                DISPLAY_CATHODE <= "01001001";
            when 6 =>
                DISPLAY_CATHODE <= "11000001";

```



```

when 7 =>
    DISPLAY_CATHODE <= "00011111";
when 8 =>
    DISPLAY_CATHODE <= "00000001";
when 9 =>
    DISPLAY_CATHODE <= "00011001";
when others =>
    DISPLAY_CATHODE <= "11111110";
end case;
when "11" =>
    DISPLAY_ANODE <= "11111110";--DISPLAY ONEs Milliseconds
case (ONE_MS) is
when 0 =>
    DISPLAY_CATHODE <= "00000011";
when 1 =>
    DISPLAY_CATHODE <= "10011111";
when 2 =>
    DISPLAY_CATHODE <= "00100101";
when 3 =>
    DISPLAY_CATHODE <= "00001101";
when 4 =>
    DISPLAY_CATHODE <= "10011001";
when 5 =>
    DISPLAY_CATHODE<= "01001001";
when 6 =>
    DISPLAY_CATHODE <= "11000001";
when 7 =>
    DISPLAY_CATHODE <= "00011111";
when 8 =>
    DISPLAY_CATHODE <= "00000001";
when 9 =>
    DISPLAY_CATHODE <= "00011001";
when others =>
    DISPLAY_CATHODE <= "11111110";
end case;
when others =>
    DISPLAY_ANODE <= "11110000";

```

```

        DISPLAY_CATHODE <= "11111110";
    end case;

    SEGMENTS := SEGMENTS + 1;

    elsif SYS_STATE = "000" then--Display during the error state.

        DISPLAY_ANODE <= "00000000";

        DISPLAY_CATHODE <= "00011001";

    end if;

end if;

end process;

end BEHAVOIRAL;

```

### 2.1.3 RANDON GENERATOR BLOCK

```

-----

-- Project Name: FPGA Coursework
-- Group: Flip Flops
-- Create Date: 1.08.2023 10:31:23
-- Module Name: RANDOM NUMBER GENERATOR
-- Target Devices: Nexys-4-DDR
-- Additional Comments:
-----

```

```

library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

entity RANDOM_NUMBER_GENERATOR is
    port (
        RAND1:  out std_logic_vector(3 downto 0); --4-bit output for the random number
        CLOCK:  in std_logic -- Input clock signals
    );
end RANDOM_NUMBER_GENERATOR;

```

architecture BEHAVIORAL of RANDOM\_NUMBER\_GENERATOR is

```

    signal LFSR_REG1:  std_logic_vector(3 downto 0) := "1010"; -- 4-bit signal for the first LFSR (Linear
Feedback Shift Register)

```

```

signal LFSR_REG2: std_logic_vector(3 downto 0) := "1010"; -- 4-bit signal for the second LFSR
signal RES :    std_logic; -- Single bit signal for the feedback XOR result
signal CLOCK1:  std_logic; -- Internal signal to hold the CLOCK value

begin

  process(CLOCK1, LFSR_REG1, LFSR_REG2)
  begin
    if rising_edge(CLOCK1) then
      LFSR_REG1 <= LFSR_REG2; -- Shift the value of LFSR_REG2 into LFSR_REG1 on the rising edge of
CLOCK1
    end if;
  end process;

  RAND1 <= LFSR_REG1;          -- Output the value of LFSR_REG1 as the random number

  CLOCK1 <= CLOCK;            --Assign the value of the input CLOCK signal to the internal signal
CLOCK1

  RES <= LFSR_REG1(1) xor LFSR_REG1(0);    -- Perform XOR operation between the two least
significant bits of LFSR_REG1 and store the result in RES

  LFSR_REG2 <= RES & LFSR_REG1(3 downto 1); -- Concatenate the value of RTN with the three most
significant bits of LFSR_REG1, and store the result in LFSR_REG2

      -- This line effectively performs the feedback and shift operations of the
LFSR to generate the next random value.

end BEHAVIORAL;

```

#### 2.1.4 LATCH

```

-----

-- Project Name: FPGA Coursework
-- Group: Flip Flops
-- Create Date: 31.07.2023 12:21:22
-- Module Name: 4- BIT LATCH
-- Target Devices: Nexys-4-DDR
-- Additional Comments:
-----

```

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

```

entity LATCH is

```
    port (  
        RAND_IN: in std_logic_vector(3 downto 0); --4-bit Random Number input for the data to be  
        latched  
        C: in std_logic; --1 bit control  
        RAND_OUT: out std_logic_vector(3 downto 0) -- 4-bit output for the latched data  
    );  
end LATCH;
```

architecture STRUCTURE of LATCH is -- Architecture of 4-bit Latch

```
begin  
    process(RAND_IN)  
    begin  
        if (C='1') then          -- Only perform the latching operation when control is high.  
            if (RAND_IN = "0001") then    --Check if the input data equals "0001"  
                RAND_OUT <= "0010";    -- If the input data is 1, set the output data to 2 (latching the new  
value)  
            else  
                RAND_OUT(0)<=RAND_IN (0); -- Latch the individual bits of Random Number input data to  
the corresponding bits of the output data  
                RAND_OUT(1)<=RAND_IN (1);  
                RAND_OUT(2)<=RAND_IN (2);  
                RAND_OUT(3)<=RAND_IN (3);  
            end if;  
        end if;          -- If the C is low, the latching operation is skipped and the output data  
remains unchanged.  
    end process;  
end STRUCTURE;
```

## 2.1.5 FOUR-BIT COMPARATOR

-----  
-- Project Name: FPGA Coursework

```

-- Group: Flip Flops
-- Create Date: 25.07.2023 13:23:38
-- Module Name: 4 BIT COMPARATOR
-- Target Devices: Nexys-4-DDR
-- Additional Comments:
-----

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity COMPARATOR_4_BIT is
    port (
        NUM1: in std_logic_vector(3 downto 0);-- 4-bit input (Counter Output)
        NUM2: in std_logic_vector(3 downto 0);-- 4-bit input (Random Number Output)
        CLK2: in std_logic; -- Clock input
        EN: in std_logic; -- Enable input
        COMP: out std_logic -- Output: '1' if NUM1 >= NUM2, '0' otherwise
    );
end COMPARATOR_4_BIT;

architecture BEHAVIORAL of COMPARATOR_4_BIT is
begin
    process(NUM1,NUM2,CLK2,EN)
    begin
        if (EN='1') then -- Check if the enable signal is asserted
            COMP <= '0'; -- If enable is '1', set the output COMP to '0'
            elsif (NUM1 >= NUM2) then-- Check if NUM1 is greater than or equal to NUM2
                COMP <= '1'; -- If true, set the output COMP to '1'
            else
                COMP <= '0'; -- Otherwise, set the output COMP to '0'
            end if;
        end process;
    end BEHAVIORAL;

```

## 2.1.6 FOUR-BIT COUNTER

```
-----  
-- Project Name: FPGA Coursework  
-- Group: Flip Flops  
-- Create Date: 27.07.2023 18:10:12  
-- Module Name: 4 BIT COUNTER  
-- Target Devices: Nexys-4-DDR  
-- Additional Comments:  
-----
```

```
library IEEE;  
use IEEE.STD_LOGIC_1164.ALL;  
use IEEE.STD_LOGIC_ARITH.all;  
use IEEE.STD_LOGIC_UNSIGNED.all;  
  
entity FOUR_BIT_COUNTER is  
    port (  
        RESET: in STD_LOGIC;-- Reset input  
        CLK_1HZ: in STD_LOGIC;-- Clock input  
        C_OUT: out STD_LOGIC_VECTOR ( 3 downto 0 )-- 4-bit output  
    );  
end FOUR_BIT_COUNTER;  
  
architecture BEHAVIORAL of FOUR_BIT_COUNTER is  
  
    signal COUNT_REG : std_logic_vector(0 to 3);-- Internal counter register  
  
begin  
    process(RESET,CLK_1HZ)-- Process to update the counter  
    begin  
        if (RESET = '1')-- Check if reset is High  
        then COUNT_REG <= "0000";-- Reset the counter to 0  
        elsif (CLK_1HZ'event and CLK_1HZ = '1')  
        then COUNT_REG <= COUNT_REG + 1;-- Increment the counter on the rising edge of the clock  
        end if;  
    end process;  
end architecture;
```

```

        C_OUT <= COUNT_REG; -- Output the 4-bit counter value

end BEHAVIORAL;

```

### 2.1.7 COUNTER MAPPING

```

-----

-- Project Name: FPGA Coursework
-- Group: Flip Flops
-- Create Date: 31.07.2023 13:22:22
-- Module Name: COUNTER BLOCK
-- Target Devices: Nexys-4-DDR
-- Additional Comments:
-----

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

library UNISIM;
use UNISIM.VCOMPONENTS.ALL;

entity COUNTER_MAPPING is
    port (
        CLK_1HZ: in STD_LOGIC;
        RESET: in STD_LOGIC;
        C_OUT: out STD_LOGIC_VECTOR ( 3 downto 0 )
    );
end COUNTER_MAPPING;

architecture BEHAVIORAL of COUNTER_MAPPING is

    component FOUR_BIT_COUNTER is
        port (
            RESET: in STD_LOGIC;-- Reset input
            CLK_1HZ: in STD_LOGIC;-- Clock input
            C_OUT: out STD_LOGIC_VECTOR ( 3 downto 0 )-- 4-bit output
        );
    end component FOUR_BIT_COUNTER;

```

```
begin
```

```
COUNTER_MAP: component FOUR_BIT_COUNTER port map (CLK_1HZ => CLK_1HZ, C_OUT(3 downto 0) => C_OUT(3 downto 0), RESET=> RESET);--Entity and component port mapping
```

```
end BEHAVIORAL;
```

## 2.1.8 COMPARATOR INTERFACE

```
-- Project Name: FPGA Coursework
```

```
-- Group: Flip Flops
```

```
-- Create Date: 25.07.2023 14:30:12
```

```
-- Module Name: COMPARATOR_INTERFACE
```

```
-- Target Devices: Nexys-4-DDR
```

```
-- Additional Comments:
```

```
library IEEE;
```

```
use IEEE.STD_LOGIC_1164.ALL;
```

```
entity COMPARATOR_INTERFACE is
```

```
Port (
```

```
    CLOCK_SECS: in std_logic;
```

```
    CON: in std_logic;
```

```
    RANDOM_NUMBER: in std_logic_vector(3 downto 0);
```

```
    COUNT_T: out std_logic
```

```
);
```

```
end COMPARATOR_INTERFACE;
```

```
architecture BEHAVIORAL of COMPARATOR_INTERFACE is
```

```
component RANDOM_NUMBER_GENERATOR is
```

```
port (
```

```
    RAND1: out std_logic_vector(3 downto 0);--4-bit output for the random number
```

```
    CLOCK: in std_logic -- Input clock signal
```



```

);

end component;

component LATCH is
    port (
        RAND_IN: in std_logic_vector(3 downto 0); --4-bit Random Number input for the data to be latched
        C:      in std_logic;--1 bit control
        RAND_OUT: out std_logic_vector(3 downto 0)-- 4-bit output for the latched data
    );
end component;

component COMPARATOR_4_BIT is
    port (
        NUM1: in std_logic_vector(3 downto 0);-- 4-bit input (Counter Output)
        NUM2: in std_logic_vector(3 downto 0);-- 4-bit input (Random Number Output)
        CLK2: in std_logic; -- Clock input
        EN:   in std_logic; -- Enable input
        COMP: out std_logic -- Output: '1' if NUM1 >= NUM2, '0' otherwise
    );
end component;

component COUNTER_MAPPING is
    port (
        CLK_1HZ: in STD_LOGIC;
        RESET:   in STD_LOGIC;
        C_OUT:   out STD_LOGIC_VECTOR ( 3 downto 0 )
    );
end component;

signal RAND_OUT_TO_NUM2: std_logic_vector(3 downto 0);
signal C_OUT_TO_NUM1: std_logic_vector(3 downto 0);

begin

COMPARATOR_MAP: COMPARATOR_4_BIT port map(NUM1=>C_OUT_TO_NUM1, NUM2=>RAND_OUT_TO_NUM2,
COMP=>COUNT_T, CLK2=>CLOCK_SECS, EN=>CON);

```

```
COUNTER_MAP: COUNTER_MAPPING port map(CLK_1HZ=>CLOCK_SECS, C_OUT=>C_OUT_TO_NUM1, RESET=>CON);
```

```
COMPARE_MAP: LATCH port map(RAND_IN=>RANDOM_NUMBER, C=>CON, RAND_OUT=>RAND_OUT_TO_NUM2);
```

```
end BEHAVIORAL;
```

## 2.1.9 BUTTON DEBOUNCE

```
-- Project Name: FPGA Coursework
```

```
-- Group: Flip Flops
```

```
-- Create Date: 3.08.2023 12:34:23
```

```
-- Module Name: BUTTON DEBOUNCER
```

```
-- Target Devices: Nexys-4-DDR
```

```
-- Additional Comments:
```

```
library IEEE;
```

```
use IEEE.STD_LOGIC_1164.ALL;
```

```
entity BUTTON_DEBOUNCER is
```

```
    Port (
```

```
        CLK5      : in STD_LOGIC;
```

```
        BTN_IN     : in STD_LOGIC;
```

```
        BTN_OUT    : out STD_LOGIC
```

```
    );
```

```
end BUTTON_DEBOUNCER ;
```

```
architecture BEHAVIORAL of BUTTON_DEBOUNCER is
```

```
    constant DEBOUNCE_CYCLES : integer := 5;-- Debounce Cycles
```

```
    signal DB_COUNTER : integer range 0 to DEBOUNCE_CYCLES;
```

```
    signal BTN_STABLE : STD_LOGIC := '0';
```

```
begin
```

```
    process(CLK5)
```

```
    begin
```

```
        if rising_edge(CLK5) then
```

```

    if BTN_IN = '1' then
        if DB_COUNTER < DEBOUNCE_CYCLES then
            DB_COUNTER <= DB_COUNTER + 1;
        else
            BTN_STABLE <= '1';
        end if;
    else
        DB_COUNTER <= 0;
        BTN_STABLE <= '0';
    end if;
end if;

end process;

BTN_OUT <= BTN_STABLE;
end BEHAVIORAL;

```

### 2.1.10 REACTION TIME

```

-- Project Name: FPGA Coursework
-- Group: Flip Flops
-- Create Date: 1.08.2023 10:31:23
-- Module Name: REACTION TIME
-- Target Devices: Nexys-4-DDR
-- Additional Comments:

```

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity REACTION_TIME is
    Port (CLOCK_MILLI : in std_logic; -- 1KHz Clock Input
          STOP       : in std_logic; --STOP BUTTON
          CLEAR      : in std_logic; --CLEAR BUTTON
          STATE      : in std_logic_vector(2 downto 0);--System State
          ONE        : out integer; -- 1 MILLISECOND
          TEN        : out integer; -- 10 MILLISECOND
          HUNDRED    : out integer; -- 100 MILLISECOND

```

```

        THOUSAND      : out integer;-- 1000 MILLISECOND/ 1 SECOND

        TIMER_STATE   : out std_logic  -- TIMER STATE

    );

end REACTION_TIME;

architecture BEHAVIORAL of REACTION_TIME is

begin

    TIMER_MILLISECONDS: process (CLOCK_MILLI, STOP, CLEAR, STATE)-- count state depends on this clock
    variable ONE_MS : integer := 0;
    variable TEN_MS : integer := 0;
    variable HUNDRED_MS : integer := 0;
    variable THOUSAND_MS : integer := 0;

    begin
        if (CLEAR='1') then -- IF USER RESETS THE TIMER
            ONE_MS := 0;
            TEN_MS := 0;
            HUNDRED_MS := 0;
            THOUSAND_MS := 0;
            ONE <= 0;
            TEN <= 0;
            HUNDRED <= 0;
            THOUSAND <= 0;
            TIMER_STATE <= '0';
        elsif rising_edge(CLOCK_MILLI) then
            if (STATE= "010") then -- Hold State
                if (STOP='1') then --User terminates TIMER by pressing stop button
                    TIMER_STATE <= '1'; -- Timer HOLD State when user presses stop.
                else
                    -- Stop not pressed by user
                    ONE_MS := ONE_MS + 1;--count in milliconds less than 10
                if ONE_MS = 9 then --Counts reaches 10 milliseconds
                    TEN_MS := TEN_MS + 1;--count in 10 milliseconds when count greater than 10ms
                    ONE_MS := 0;
                if TEN_MS = 9 then --Count reaches 100 milliseconds
                    HUNDRED_MS := HUNDRED_MS + 1;---Count in 100 milliseconds
                    TEN_MS := 0;

```

```

        if HUNDRED_MS = 9 then----Count reaches 1000 milliseconds
            THOUSAND_MS := THOUSAND_MS + 1;--Count in 1000 milliseconds
            HUNDRED_MS := 0;
            TIMER_STATE <= '1'; --Timer HOLD State when counter reaches 1000 milliseconds
        end if;
    end if;
end if;
end if;
end if;
end if;
end if;

ONE    <= ONE_MS;
TEN    <= TEN_MS;
HUNDRED <= HUNDRED_MS;
THOUSAND <= THOUSAND_MS;
end process;

end BEHAVIORAL;

```

### 2.1.11 HUMAN REACTION (MAIN)

```

-- Project Name: FPGA Coursework
-- Group: Flip Flops
-- Create Date: 2.08.2023 18:30:22
-- Module Name: MAIN
-- Target Devices: Nexys-4-DDR
-- Additional Comments:

```

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

```

```

entity HUMAN_REACTION is
    port (

```

```

START:      in std_logic;          -- START Button Input
STOP:       in std_logic;          -- START Button Input
CLEAR:      in std_logic;          -- START Button Input
CLOCK_INPUT: in std_logic;          -- System 100MHz Clock Input
STIMULUS:    out std_logic_vector(1 downto 0); -- LED Stimulus Output
SEGMENT_CATHODE: out std_logic_vector(7 downto 0); -- Output for Displaying Characters on 7
Segment Display
SEGMENT_ANODE: out std_logic_vector(7 downto 0) -- Output for Turning on 7 Segment Display
);

```

```

end HUMAN_REACTION;

```

architecture BEHAVIORAL of HUMAN\_REACTION is

```

signal BTN_START_DEBOUNCED : STD_LOGIC := '0';
signal BTN_STOP_DEBOUNCED : STD_LOGIC := '0';
signal BTN_CLEAR_DEBOUNCED : STD_LOGIC := '0';

signal BTN_START_STABLE : STD_LOGIC; -- Instantiate button debounce circuits
signal BTN_STOP_STABLE : STD_LOGIC;
signal BTN_CLEAR_STABLE : STD_LOGIC;

```

Component BUTTON\_DEBOUNCER is

```

Port (
    CLK5      : in STD_LOGIC;
    BTN_IN    : in STD_LOGIC;
    BTN_OUT   : out STD_LOGIC
);

```

```

end component;

```

component COMPARATOR\_INTERFACE is

```
Port (  
    CLOCK_SECS: in std_logic;  
    CON:        in std_logic;  
    RANDOM_NUMBER: in std_logic_vector(3 downto 0);  
    COUNT_T:    out std_logic  
);
```

end component;

component RANDOM\_NUMBER\_GENERATOR is

```
port (  
    RAND1: out std_logic_vector(3 downto 0); --4-bit output for the random number  
    CLOCK: in std_logic -- Input clock signals  
);
```

end component;

component CLOCK\_DIVIDER is

```
port (  
    CLOCK_INPUT: in std_logic; -- 100MHz Input Clock Signal  
    CLOCK_DISP: out std_logic; -- 2KHz Clock output for refreshing display every 0.5ms  
    CLOCK_MILLI: out std_logic; -- 1KHz Clock output for Counting in milli seconds  
    CLOCK_SEC: out std_logic -- 1Hz Clock output for counting in seconds  
);
```

end component;

component SEVEN\_SEGMENT\_DISPLAY is

```
Port (  
    CLOCK_DISP : in std_logic;  
    SYS_STATE : in std_logic_vector(2 downto 0);  
    ONE_MS : in integer;  
    TEN_MS : in integer;
```

```

        HUNDRED_MS      : in integer;

        THOUSAND_MS     : in integer;

        DISPLAY_ANODE    : out std_logic_vector(7 downto 0);

        DISPLAY_CATHODE  : out std_logic_vector(7 downto 0)

    );

end component;

component REACTION_TIME is
    Port (CLOCK_MILLI : in std_logic; -- 1KHz Clock Input

        STOP      : in std_logic; --STOP BUTTON

        CLEAR     : in std_logic; --CLEAR BUTTON

        STATE     : in std_logic_vector(2 downto 0); --System State

        ONE       : out integer; -- 1 MILLISECOND

        TEN       : out integer; -- 10 MILLISECOND

        HUNDRED   : out integer; -- 100 MILLISECOND

        THOUSAND  : out integer; -- 1000 MILLISECOND/ 1 SECOND

        TIMER_STATE : out std_logic -- TIMER STATE

    );

end component;

```

```

type STAGE is (INITIALIZE, RANDOM_TIME, COUNTING, ERROR, HOLD);

signal STATE : STAGE := INITIALIZE;

signal STATE_WRAPPER:      std_logic_vector(2 downto 0);

signal RN:                  std_logic_vector(3 downto 0);

signal CLOCK_IN_SEC:        std_logic;

signal CLOCK_IN_MILLISECONDS:  std_logic;

signal CLOCK_REFRESH_RATE:   std_logic;

signal CLOCK_SYS:           std_logic;

signal CONTROL:             std_logic;

```



```

signal TRIGGER_COUNT:      std_logic;
signal RESPONSE:           std_logic_vector(11 downto 0);
signal COUNT_STATE :      std_logic;
signal ONE_SIG:            integer;
signal TEN_SIG:            integer;
signal HUNDRED_SIG:        integer;
signal THOUSAND_SIG:       integer;

begin

CLOCK_SYS <= CLOCK_INPUT;

BUTTONDEBOUNCE_START: BUTTON_DEBOUNCER port map (CLK5 => CLOCK_SYS, BTN_IN => START,
BTN_OUT => BTN_START_STABLE);

BUTTONDEBOUNCE_STOP: BUTTON_DEBOUNCER port map (CLK5 => CLOCK_SYS, BTN_IN => STOP,
BTN_OUT => BTN_STOP_STABLE);

BUTTONDEBOUNCE_CLEAR: BUTTON_DEBOUNCER port map(CLK5 => CLOCK_SYS, BTN_IN => CLEAR,
BTN_OUT => BTN_CLEAR_STABLE);

COMPARE_BLOCK: COMPARATOR_INTERFACE port map(CLOCK_SECS => CLOCK_IN_SEC,
RANDOM_NUMBER =>RN, CON=>CONTROL, COUNT_T=>TRIGGER_COUNT);

RANDOM_TIME_BLOCK: RANDOM_NUMBER_GENERATOR port map (RAND1=>RN,
CLOCK=>CLOCK_SYS);

CLOCK_DIVIDER_BLOCK: CLOCK_DIVIDER port map(CLOCK_INPUT=>CLOCK_SYS,
CLOCK_MILLI=>CLOCK_IN_MILLISECONDS, CLOCK_SEC=>CLOCK_IN_SEC,
CLOCK_DISP=>CLOCK_REFRESH_RATE);

DISPLAY_BLOCK: SEVEN_SEGMENT_DISPLAY port map (CLOCK_DISP=>CLOCK_REFRESH_RATE,
ONE_MS=>ONE_SIG,          TEN_MS=>TEN_SIG,          HUNDRED_MS=>HUNDRED_SIG,

```

```

THOUSAND_MS=>THOUSAND_SIG,                                SYS_STATE=>STATE_WRAPPER,
DISPLAY_CATHODE=>SEGMENT_CATHODE, DISPLAY_ANODE=>SEGMENT_ANODE);

```

```

REACTION_TIME_BLOCK: REACTION_TIME port map (CLOCK_MILLI =>CLOCK_IN_MILLISECONDS,
CLEAR  =>  BTN_CLEAR_STABLE, STOP  =>  BTN_STOP_STABLE, STATE  =>  STATE_WRAPPER,
ONE=>ONE_SIG, TEN=>TEN_SIG, HUNDRED=>HUNDRED_SIG, THOUSAND=>THOUSAND_SIG,
TIMER_STATE => COUNT_STATE);

```

```

STATES: process (BTN_START_STABLE, BTN_STOP_STABLE, BTN_CLEAR_STABLE, CLOCK_SYS)

```

```

begin

```

```

    if (BTN_CLEAR_STABLE ='1') then

```

```

        STATE <= INITIALIZE;

```

```

        STIMULUS <= "00";

```

```

    elsif rising_edge(CLOCK_SYS)then

```

```

        case STATE is

```

```

            when INITIALIZE =>

```

```

                CONTROL <= '1';

```

```

                STATE_WRAPPER <= "100";

```

```

            if (START='1') then

```

```

                STATE <=RANDOM_TIME;

```

```

            end if;

```

```

            when RANDOM_TIME =>

```

```

                STATE_WRAPPER <= "101";

```

```

                CONTROL<= '0';

```

```

            if (BTN_STOP_STABLE='1') then

```

```

                STATE <= ERROR;

```

```

    elsif (BTN_CLEAR_STABLE='1') then
        STATE <= INITIALIZE;

    elsif (TRIGGER_COUNT='1') then
        STATE <= COUNTING;
    end if;

when COUNTING =>
    STIMULUS <= "11"; --Turn on 2x green LEDS for STIMULUS
    STATE_WRAPPER <= "111";
    CONTROL <= '1';
    if (BTN_STOP_STABLE='1') then
        STATE <= HOLD;
    elsif (BTN_CLEAR_STABLE='1') then
        STATE <= INITIALIZE;
    elsif (TRIGGER_COUNT = '1') then
        STATE <= HOLD;
    end if;

when ERROR =>
    STATE_WRAPPER <= "000";
    CONTROL <= '1';
    if (BTN_CLEAR_STABLE='1') then
        STATE <= INITIALIZE;
    end if;

when HOLD =>
    STATE_WRAPPER <= "011";
    STIMULUS <= "00";
    CONTROL <= '1';

```

```

        if (BTN_CLEAR_STABLE='1') then
            STATE <= INITIALIZE;
        end if;

        when others =>
            STATE <= INITIALIZE;
            CONTROL <= '1';
        end case;
    end if;
end process;

end BEHAVIORAL;

```

### 2.2.1 CLOCK DIVIDER TESTBENCH

```

-----
-- Project Name: FPGA Coursework
-- Group: Flip Flops
-- Create Date: 25.07.2023 14:30:12
-- Module Name: CLOCK DIVIDER TEST BENCH
-- Target Devices: Nexys-4-DDR
-- Additional Comments:
-----

```

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

```

```

entity clock_divider_tb is
end clock_divider_tb;

architecture TESTING of clock_divider_tb is

constant CLOCK_PERIOD_tb : time := 10 ns;

component CLOCK_DIVIDER
    port (
        CLOCK_INPUT: in std_logic;
        CLOCK_DISP: out std_logic;
        CLOCK_MILLI: out std_logic;
        CLOCK_SEC: out std_logic
    );
end component;

signal CLOCK_INPUT_TB : std_logic := '0';
signal CLOCK_DISP_TB : std_logic;
signal CLOCK_MILLI_TB : std_logic;
signal CLOCK_SEC_TB : std_logic;

begin

DUT: CLOCK_DIVIDER port map ( CLOCK_INPUT => CLOCK_INPUT_TB, CLOCK_DISP => CLOCK_DISP_TB, CLOCK_MILLI =>
CLOCK_MILLI_TB, CLOCK_SEC => CLOCK_SEC_TB);

CLK_GEN: process
begin

    CLOCK_INPUT_TB <= '0';-- Initial value
    wait for CLOCK_PERIOD_TB/2;-- Wait half of the clock period
    CLOCK_INPUT_TB <= '1';-- Toggle the clock
    wait for CLOCK_PERIOD_TB/2;-- Wait for the rest of the clock period

end process;

```

```
end TESTING;
```

## 2.2.2 4-BIT COMPARATOR TEST BENCH

```
-----  
-- Project Name: FPGA Coursework  
-- Group: Flip Flops  
-- Create Date: 04.08.2023 05:38:23  
-- Module Name: 4 BIT COMPARATOR TEST BENCH  
-- Target Devices: Nexys-4-DDR  
-- Additional Comments:  
-----
```

```
library IEEE;  
use IEEE.STD_LOGIC_1164.ALL;  
use IEEE.STD_LOGIC_ARITH.ALL;  
use IEEE.STD_LOGIC_UNSIGNED.ALL;
```

```
entity COMPARATOR_TB is  
end COMPARATOR_TB;
```

```
architecture TESTING of COMPARATOR_TB is
```

```
component COMPARATOR_4_BIT -- Component Declaration for the COMPARATOR_4_BIT entity
```

```
port (  
    NUM1: in std_logic_vector(3 downto 0);  
    NUM2: in std_logic_vector(3 downto 0);  
    CLK2: in std_logic;  
    EN:   in std_logic;  
    COMP: out std_logic  
);
```

```
end component;
```

```
signal NUM1_TB : std_logic_vector(3 downto 0) := "0000"; -- Signals for the testbench  
signal NUM2_TB : std_logic_vector(3 downto 0) := "0000";  
signal CLK2_TB : std_logic := '0';
```

```

signal EN_TB : std_logic := '0';
signal COMP_TB : std_logic;

constant CLK_PERIOD: time := 10 ns;-- Define the clock period for 100MHz

begin

CLOCK_PROCESS: process

begin

    while now < 1000 ns loop -- Run the simulation for 1000 ns

        CLK2_TB <= '0';

        wait for CLK_PERIOD/2;

        CLK2_TB <= '1';

        wait for CLK_PERIOD/2;

    end loop;

    wait;

end process;


COMPARATOR_PROCESS: process

begin


    EN_TB <= '0';    --Disable initially

    NUM1_TB <= "0000";

    NUM2_TB <= "0000";


    wait for 100 ns; -- Wait for a few clock cycles before enabling the comparator


    num1_tb <= "0010"; -- Test Case 1: NUM1=NUM2

    num2_tb <= "0010";

    en_tb <= '1';

    wait for 100 ns;


    num1_tb <= "1010"; -- Test Case 2: NUM1 > NUM2

    num2_tb <= "0010";

    en_tb <= '1';

```

```

wait for 100 ns;

num1_tb <= "0001"; -- Test Case 3: NUM1 < NUM2
num2_tb <= "0010";
en_tb <= '1';
wait for 100 ns;

en_tb <= '0'; -- Disable the comparator
wait for 100 ns;

wait;
end process;

COMPARATOR_INIT: COMPARATOR_4_BIT port map (NUM1 => NUM1_TB, NUM2 => NUM2_TB, CLK2 => CLK2_TB, EN =>
EN_TB, COMP => COMP_TB);

end TESTING;

```

### 2.2.3 4-BIT COUNTER TEST BENCH

```

-----
-- Project Name: FPGA Coursework
-- Group: Flip Flops
-- Create Date: 04.08.2023 05:16:11
-- Module Name: 4 BIT COUNTER TEST BENCH
-- Target Devices: Nexys-4-DDR
-- Additional Comments:
-----

```

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity FOUR_BIT_COUNTER_TB is

```



```
end FOUR_BIT_COUNTER_TB;
```

architecture TESTING of FOUR\_BIT\_COUNTER\_TB is

component FOUR\_BIT\_COUNTER -- Component Declaration for the FOUR\_BIT\_COUNTER entity

```
    port (  
        RESET:  in STD_LOGIC;  
        CLK_1HZ:  in STD_LOGIC;  
        C_OUT:   out STD_LOGIC_VECTOR ( 3 downto 0 )  
    );
```

```
end component;
```

```
signal CLK_1HZ_TB : std_logic := '0';-- Signals for the testbench
```

```
signal RESET_TB: std_logic := '0';
```

```
signal C_OUT_TB : std_logic_vector(3 downto 0);
```

```
constant CLK_PERIOD : time := 1 us;
```

```
begin
```

```
CLOCK_GEN: process
```

```
    begin
```

```
        while now < 1000 us loop
```

```
            CLK_1HZ_TB <= '0';
```

```
            wait for CLK_PERIOD/2;
```

```
            CLK_1HZ_TB <= '1';
```

```
            wait for CLK_PERIOD/2;
```

```
        end loop;
```

```
        wait;
```

```
    end process;
```

```
COUNTER_PROCESS:
```

```
process
```

```
    begin
```

```
        RESET_TB <= '1'; -- Reset the counter initially
```

```
wait for 5 us; -- Wait for some time  
RESET_TB <= '0'; -- Release reset  
wait for 100 us;-- Wait for some time
```

```
wait for 500 us;-- Test the Counter output for 500us
```

```
wait;  
end process;
```

```
COUNTER_INIT: FOUR_BIT_COUNTER port map (RESET => RESET_TB, CLK_1HZ => CLK_1HZ_TB, C_OUT => C_OUT_TB);
```

```
end TESTING;
```

## 2.2.4 SEVEN SEGMENT DISPLAY TEST BENCH

```
-----  
-- Project Name: FPGA Coursework  
-- Group: Flip Flops  
-- Create Date: 04.08.2023 06:34:10  
-- Module Name: DISPLAY TEST BENCH  
-- Target Devices: Nexys-4-DDR  
-- Additional Comments:  
-----
```

```
library IEEE;  
use IEEE.STD_LOGIC_1164.ALL;  
use IEEE.NUMERIC_STD.ALL;
```

```
entity SEVEN_SEGMENT_DISPLAY_TB is
end SEVEN_SEGMENT_DISPLAY_TB;
```

architecture TESTING of SEVEN\_SEGMENT\_DISPLAY\_tb is

```
component SEVEN_SEGMENT_DISPLAY
```

```
    Port (
        CLOCK_DISP: in std_logic;
        ONE_MS: in integer;
        TEN_MS: in integer;
        HUNDRED_MS: in integer;
        THOUSAND_MS: in integer;
        SYS_STATE: in std_logic_vector(2 downto 0);
        DISPLAY_ANODE: out std_logic_vector(7 downto 0);
        DISPLAY_CATHODE: out std_logic_vector(7 downto 0)
    );
```

```
end component;
```

```
signal CLOCK_DISP_TB : std_logic := '0';
signal ONE_MS_TB : integer := 0;
signal TEN_MS_TB : integer := 0;
signal HUNDRED_MS_TB : integer := 0;
signal THOUSAND_MS_TB : integer := 0;
signal SYS_STATE_TB : std_logic_vector(2 downto 0) := "000";
signal DISPLAY_ANODE_TB : std_logic_vector(7 downto 0);
signal DISPLAY_CATHODE_TB : std_logic_vector(7 downto 0);
```

```
constant CLK_PERIOD : time := 1 us;-- Define the clock period
```

```
begin
```

```
CLOCK_GEN: process
```

```
begin
    while now < 20000 us loop -- Run the simulation for 20 ms
        CLOCK_DISP_TB <= '0';
        wait for CLK_PERIOD/2;
        CLOCK_DISP_TB <= '1';
        wait for CLK_PERIOD/2;
```

```
    end loop;

    wait;

end process;
```

```
DISPLAY_PROCESS: process
```

```
begin
```

```
    SYS_STATE_TB <= "100";-- Test Case 1: SYS_STATE = "100" (Initialization State)
```

```
    wait for 100 us; -- Wait before testing next state
```

```
    SYS_STATE_TB <= "101";-- Test Case 2: SYS_STATE = "101" (Random Wait Interlock State)
```

```
    wait for 100 us;-- Wait before testing next state
```

```
    SYS_STATE_TB <= "111";-- Test Case 3: SYS_STATE = "111" (Counting State), and display various times
```

```
    ONE_MS_TB <= 5;
```

```
    TEN_MS_TB <= 2;
```

```
    HUNDRED_MS_TB <= 1;
```

```
    THOUSAND_MS_TB <= 0;
```

```
    wait for 100 us;
```

```
    SYS_STATE_TB <= "011"; -- Test Case 4: SYS_STATE = "011" (Hold State), and display various times
```

```
    ONE_MS_TB <= 9;
```

```
    TEN_MS_TB <= 9;
```

```
    HUNDRED_MS_TB <= 9;
```

```
    THOUSAND_MS_TB <= 9;
```

```
    wait for 100 us;-- Wait before testing next state
```

```
    SYS_STATE_TB <= "000"; -- Test Case 5: SYS_STATE = "000" (Error State), and display an error
```

```
    wait for 100 us;
```

```

        wait;
    end process;

    DISPLAY_INIT: SEVEN_SEGMENT_DISPLAY

    port map (

        CLOCK_DISP => CLOCK_DISP_TB,

        ONE_MS => ONE_MS_TB,

        TEN_MS => TEN_MS_TB,

        HUNDRED_MS => HUNDRED_MS_TB,

        THOUSAND_MS => THOUSAND_MS_TB,

        SYS_STATE => SYS_STATE_TB,

        DISPLAY_ANODE => DISPLAY_ANODE_TB,

        DISPLAY_CATHODE => DISPLAY_CATHODE_TB

    );

end TESTING;

```

## 2.2.5 LATCH TEST BENCH

```

-----
-- Project Name: FPGA Coursework
-- Group: Flip Flops
-- Create Date: 04.08.2023 06:34:10
-- Module Name: LATCH TEST BENCH
-- Target Devices: Nexys-4-DDR
-- Additional Comments:
-----

```

```

library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

entity LATCH_TB is

end LATCH_TB;

```

architecture TESTING of LATCH\_TB is -- Component Declaration for the LATCH entity

component LATCH

port (

RAND\_IN: in std\_logic\_vector(3 downto 0);

C: in std\_logic;

RAND\_OUT: out std\_logic\_vector(3 downto 0)

);

end component;

signal CLOCK : std\_logic := '0'; -- Signals for the testbench

signal RAND\_IN\_tb : std\_logic\_vector(3 downto 0) := "0000";

signal C\_tb : std\_logic := '0';

signal RAND\_OUT\_tb : std\_logic\_vector(3 downto 0);

constant CLOCK\_PERIOD : time := 10 ns;

begin

CLK\_PROCESS: process

begin

while now < 1000 ns loop -- Run the simulation for 1000 ns

CLOCK <= '0';

wait for CLOCK\_PERIOD/2;

CLOCK <= '1';

wait for CLOCK\_PERIOD/2;

end loop;

wait;

end process;

LATCH\_PROCESS: process

begin

RAND\_IN\_tb <= "0001";-- Set initial values for the inputs

C\_tb <= '0';

```
wait for 20 ns;-- Wait for a few clock cycles before changing inputs
```

```
C_tb <= '1';-- Test Case 1: Set the control to '1' and set the input data to "0001"
```

```
RAND_IN_tb <= "0001";
```

```
wait for 20 ns;
```

```
RAND_IN_tb <= "1010";-- Test Case 2: Set the control to '1' and change the input data to "1010"
```

```
wait for 20 ns;
```

```
C_tb <= '0';-- Test Case 3: Set the control back to '0', no latching should occur
```

```
wait for 20 ns;
```

```
C_tb <= '1';-- Test Case 4: Set the control to '1' again, and change input data to "1111"
```

```
RAND_IN_tb <= "1111";
```

```
wait for 20 ns;
```

```
wait;
```

```
end process;
```

```
LATCH_INIT: LATCH -- Instantiate the LATCH entity
```

```
port map (
```

```
    RAND_IN => RAND_IN_tb,
```

```
    C => C_tb,
```

```
    RAND_OUT => RAND_OUT_tb
```

```
);
```

```
end TESTING;
```

## 2.2.6 RANDOM NUMBER GENERATOR TEST BENCH

```
-----  
-- Project Name: FPGA Coursework
```

```
-- Group: Flip Flops
```

```
-- Create Date: 3.08.2023 04:31:23
```

```
-- Module Name: RANDOM NUMBER GENERATOR TESTBENCH
```

```
-- Target Devices: Nexys-4-DDR
```

```
-- Additional Comments:  
-----
```

```
library IEEE;
```

```
use IEEE.STD_LOGIC_1164.ALL;
```

```
entity RAND_TB is
```

```
end RAND_TB;
```

```
architecture TESTING of RAND_TB is
```

```
component RANDOM_NUMBER_GENERATOR is--Component Declaration of Device under test
```

```
port (
```

```
    RAND1: out std_logic_vector(3 downto 0);--4-bit output for the random number
```

```
    CLOCK: in std_logic -- Input clock signals
```

```
);
```

```
end component;
```

```
signal CLOCK : std_logic := '0';
```

```
signal RAND1 : std_logic_vector(3 downto 0);
```

```
begin
```

```
DUT: RANDOM_NUMBER_GENERATOR port map (RAND1 => RAND1, CLOCK => CLOCK);
```

```
process
```

```
begin
```

```
    CLOCK <= '0';
```

```
    wait for 5ns;
```

```
    CLOCK <= '1';
```

```
    wait for 5ns;
```

```
end process;
```



```
end TESTING;
```

### 2.2.7 BUTTON DEBOUNCE TEST BENCH

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity BUTTON_DEBOUNCER_TB is
end BUTTON_DEBOUNCER_TB;

architecture TESTING of BUTTON_DEBOUNCER_TB is

    component BUTTON_DEBOUNCER
        Port (
            CLK5: in STD_LOGIC;
            BTN_IN: in STD_LOGIC;
            BTN_OUT: out STD_LOGIC
        );
    end component;

    signal CLK5_TB : std_logic := '0';-- Signals for the testbench
    signal BTN_IN_TB : std_logic := '0';
    signal BTN_OUT_TB : std_logic;

    constant CLK_PERIOD : time := 1 us;-- Define the clock period

    begin
    CLK_GEN: process
        begin
            while now < 10000 us loop -- Run the simulation for 10 ms
                CLK5_TB <= '0';
                wait for CLK_PERIOD/2;
                clk5_tb <= '1';
                wait for CLK_PERIOD/2;
            end loop;
            wait;
        end process;
```

```

-- Stimulus process
DEBOUNCE_PROCESS: process
begin
    BTN_IN_TB <= '0'; -- Set BTN_IN to low initially

    wait for 10 us;

    BTN_IN_TB <= '1';-- Test Case 1: Press the button, BTN_IN = '1'

    wait for 2 us;

    BTN_IN_TB <= '0';-- Test Case 2: Release the button, BTN_IN = '0'

    wait for 3 us;

    --

    BTN_IN_TB <= '1'; -- Test Case 3: Press the button again, BTN_IN = '1'

    wait for 5 us;

    --

    wait for 8 us;-- Test Case 4: Wait for debouncing, BTN_OUT should remain '0'

    wait for 5 us; -- Test Case 5: BTN_OUT should become '1' after debouncing

    wait;
end process;

BUTTON_DEBOUNCER_inst: BUTTON_DEBOUNCER
port map (
    CLK5 => clk5_tb,
    BTN_IN => BTN_IN_TB,
    BTN_OUT => BTN_OUT_TB
);
end TESTING;

```

## 2.2.8 REACTION TIME TEST BENCH

```

-----

-- Project Name: FPGA Coursework
-- Group: Flip Flops
-- Create Date: 04.08.2023 06:09:40
-- Module Name: REACTION TIME TEST BENCH
-- Target Devices: Nexys-4-DDR
-- Additional Comments:
-----

```

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity REACTION_TIME_TB is
end REACTION_TIME_TB;

architecture TESTING of REACTION_TIME_TB is

component REACTION_TIME
    Port (
        CLOCK_MILLI: in std_logic;
        STOP: in std_logic;
        CLEAR: in std_logic;
        STATE: in std_logic_vector(2 downto 0);
        ONE: out integer;
        TEN: out integer;
        HUNDRED: out integer;
        THOUSAND: out integer;
        TIMER_STATE: out std_logic
    );
end component;

signal CLOCK_MILLI_TB : std_logic := '0';
signal STOP_TB : std_logic := '0';
signal CLEAR_TB : std_logic := '0';
signal STATE_TB : std_logic_vector(2 downto 0) := "000";
signal ONE_TB : integer := 0;
signal TEN_TB : integer := 0;
signal HUNDRED_TB : integer := 0;
signal THOUSAND_TB : integer := 0;
signal TIMER_STATE_TB : std_logic := '0';

constant CLK_PERIOD : time := 1 us;-- Define the clock period

begin

```

```
CLK_PROCESS: process
begin
    while now < 20000 us loop
        CLOCK_MILLI_TB <= '0';
        wait for CLK_PERIOD/2;
        CLOCK_MILLI_TB <= '1';
        wait for CLK_PERIOD/2;
    end loop;
    wait;
end process;
```

```
REACTION_TIME_PROCESS: process
begin
```