# ENEL373:
# Reaction Timer Project

Friday Group 13

Jack Edwards

Rob Bass

Brent Smith

17/05/2024

## Contents

## Introduction:

The goal of this project was to implement a reaction timing program on a Nexus 7 development board, which contains an embedded Artix-7 FPGA, as well as several peripherals and interfaces, such as an 8 digit 7-segment display, and five push buttons.

The timer program was required to count down to count down in 3 steps, displaying a corresponding number of dots on the display at each step, before starting a timer that ran until the user presses the middle push button, at which point the measured reaction time was shown on the display.

Once the test has been successfully completed the user may either press the middle button to run the reaction time test again, or press one of the other push buttons to view statistics about the last 3 recorded reaction times, with the ‘up’ button displaying the slowest time, the ‘down’ button displaying the fastest time, and the ‘right’ button displaying the mean of the last 3 recorded times. The memory may also be cleared by pressing the ‘left’ push button.

The timer was also required to comply with either one of two additional specifications; either the timing of the test countdown should be randomized so that the user cannot predict when the timer will start, or the number of recorded times that can be stored should be user configurable.

The completed timer program successfully met all the required specifications. The countdown timer varied the time for the last step between 250ms and 2250ms, preventing the user from predicting when the timer would start. A button press too early would briefly show the word “Error” on the display before restarting the countdown process.

Times were recorded and displayed in milliseconds up to a maximum of four digits, and the maximum, minimum, and average times were displayed when the corresponding buttons were pressed, indicated by “‾‾‾‾”, “____”, and “----” respectively in the first four digits of the display.

For this project it was decided that randomizing the countdown time would be easier to implement, and that providing a more accurate test of reaction times was more useful than being able to store a larger number of reaction times.

This project was also a success in terms of becoming more familiar with working VHDL, and how FPGAs differ from more traditional microcontroller architectures and programming languages.

## Top-level block diagram:

The top-level block diagram of the final design is shown below in figure 1.

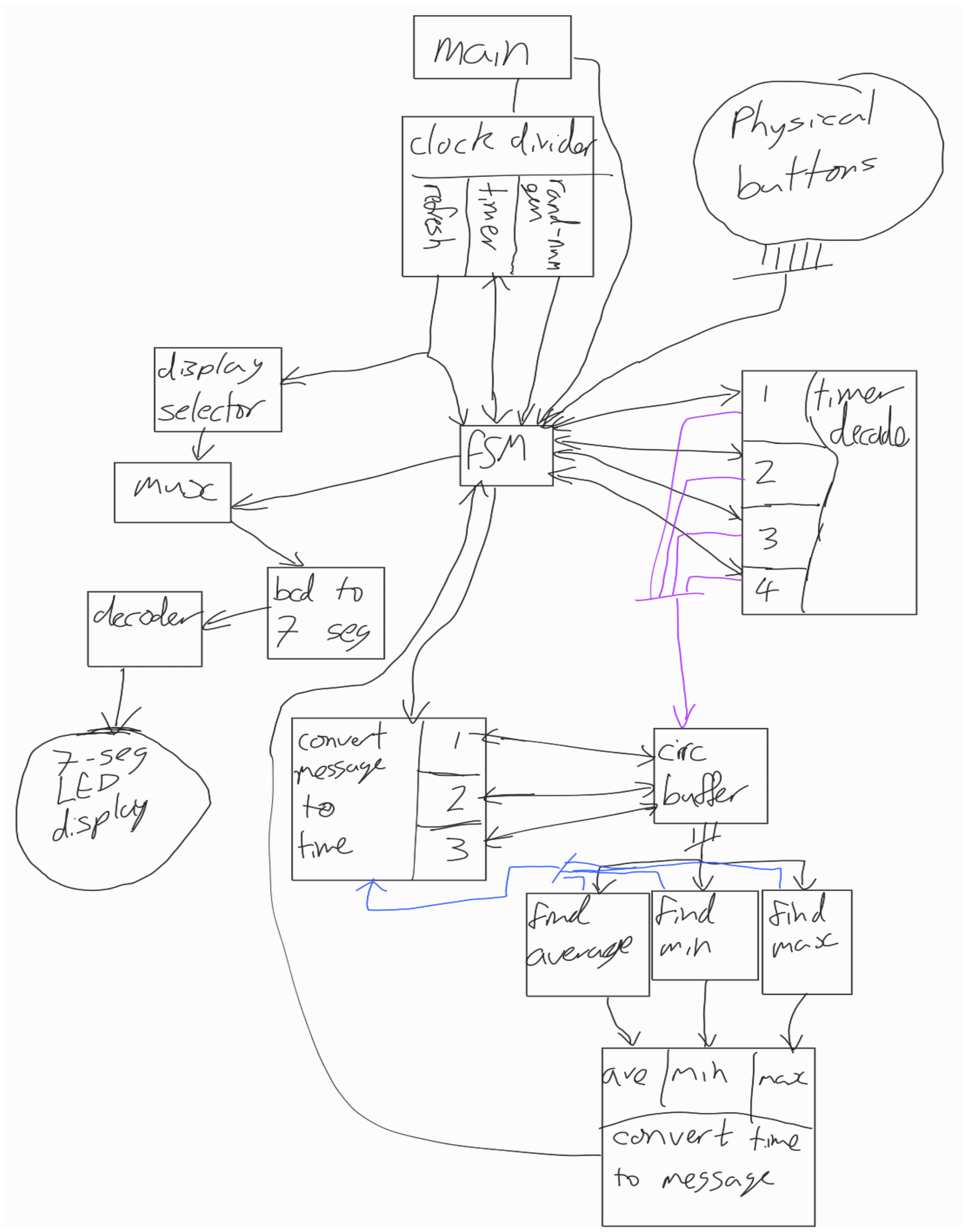


**Figure 1** Top-level block diagram of the reaction timer project.

## Expanded design summary:

The implementation of the FPGA design followed a process which considered the architectural design and the RTL programming. The sections below go into the specifics of what was considered for said section.

### Architectural design

The architectural design phase was involved taking modules that were made in previous labs to be used for this project and figuring out how the data paths would interconnect these modules as shown in Figure 1 above. The modules created were design to complete set tasks simply and only complete that task given. This can be shown by having modules like the decade timer be connected to four identical timers, to create a timer which can count from 0 to 9999, as shown in Appendix B. Each counter counted from 0 up 9, then activated a "TICK" signal and reset to 0. The first counter would increment when it received an input signal from the clock, which was divided so the clocked ticked once a millisecond. The first counter measured milliseconds. The second counter incremented on the first's "TICK", which allowed it to measure 10s of milliseconds. The third and fourth counters repeated this, counting 100s of milliseconds and 1s of seconds respectively, giving the full timer a range of 10 seconds.

The main centres of the system are MAIN and FSM (Finite State Machine). All modules are instantiated within MAIN, linking all the busses between the modules, allowing them to communicate and somewhat altering the input of some of the peripherals, so they are more easily understood by the peripherals. The FSM naturally controls what state the system is in (the states being: Warning {3, 2, 1}, Counting, Printing, Error, Display {Min, Max, Average}) and what the system does in each state, so where it gets what should be displayed from, how what should be displayed is calculated and feeding this data to the modules which actually display it. It does this via a variety of ENABLE, RESET and MESSAGE signals, sent and received from different modules (MESSAGE signals being 16 or 32 bit signals which carry what should be displayed, 4 bits per digit).

The randomiser is simply another clock divider. It counts to an upper bound based on the FPGA's clock, then resets when it reaches it. Instead of sending a signal somewhere else when it resets, it instead has its value captured when the FSM enters Warning 2. This is then multiplied by 32 and has 256 added to it, to ensure a sufficient minimum warning and range of possible warnings can occur. Whatever number is then pulled from the randomiser, is how many milliseconds Warning 3 is active for. The randomiser uses an upper bound of 31, a prime number which is conveniently the highest number which can be stored with five digits in binary (how many are used in the number selection process). The prime number means that what number is picked should not depend on how the other clocks influence the system and the fact that the number changes at 100 MHz makes it practically impossible to predict how long the random delay will be, even if it's very much a pseudo random number. It is also worth noting that only the third dot turning off is random. This is as the group decided that making all three random may reduce the effect of the randomness, as the previous delays may influence the later ones. Having them be random also defeats the purpose of them being a warning, they should tell the user to get ready and give them time to prepare, whilst only the third needs to be random, in order to prevent the user pre-empting the light turning off.

The interconnected signals were specifically chosen so certain parts of a module were only accessible by that module and not editable by others. This can be shown by the actual counter in the decade timer was only an output from the module and an internal signal was created which would then be assigned to the output after the program had run its course.

The data paths were chosen to give only very specific parts of each module which was accessible to the other modules and only the data paths which were connected to the circular buffer storage were both inputs and outputs without having an internal signal between them as shown in the top-level block diagram above.

## RTL programming

After the architectural design was considered the Register Transfer Level (RTL) programming was considered which translated the design into RTL code using VHDL in the Vivado programming software. This process involved describing the behavior and functionality of each module, specifying the data flow, and defining the control logic.

The RTL code was structured in a way that led to a high modularity and simply modules which were easy to read. We used proper naming conventions for all interconnected and internal signals helping to create easy to read code which was able to be edited and debugged in a quick manner. Comments were added to the code to explain sections which were not clearly understandable from just the signal names alone.

## Final design justification

The final design was programmed to meet all the required specification mentioned in the project brief [1]. It also was looked at how best to optimize the modularity and performance of the reaction timer. This section of the report goes into more detail about the rationale made during the implementation of the design.

### Modularity

One of the primary considerations when implementing the design was to ensure modularity. By programming the functions for different processes into multiple modules it meant that the code was easy to understand. This meant that changes to internal processes were easier and was clear what information was accessible to the module being worked on. This led to being able to implement new functions easier by creating more modules without having to redesign large parts of the program.

The modularity of the design also promoted creating modules which could be reused multiple times. The decade timer is a good example of this as it was used four times to create a timer which was able to count from 0 up to 9999 and had the capability to count higher with the addition of more decade timers as shown in Figure 4 in Appendix B. This helped with testing modules as they could be separated out and tested individually and then tested together to get a view of them working in the project.

### Performance

The performance was a critical part of all the modules related to timing as there were hard requirements [1] on the performance of the modules. As FPGAs implement what they are programmed to do in hardware, this allows all the modules to run concurrently, rather sequentially, even if individual processes within the modules are sequential. This effectively reduces the overhead delay and increases the system's performance. Furthermore, parallelization of tasks involving timing with the rest of the system was vital as the project needs accurate timing to complete its requirements as a reaction timer.

More systems in the reaction timer were made into parallel modules which were required so the performance of the overall program was sufficient. This was done to minimize latency and maximize efficiency by streamlining the data flow between modules and reducing overall time between processes completion therefore increasing system performance.

## Module testing:

A main testbench was created to help test any modules that needed to be tested as shown in Figure 2 in Appendix A. This testbench was used to help test the decade timers with the actual use case of the program. A waveform of a single decade timer is shown in Figure 3 in Appendix A where it shows the internal signals that occur when in the counting state and changing into the printing state after pressing the centre button. The waveform shows that the decade timer is properly counting the 9 from 0 and then will activate the TICK signal which will then be sent to another decade timer as the INCREMENT input also shown there. The transition to the printing state is also shown in the waveform which shows the EN signal turn off stopping the count of the decade timer.

Truthfully, this project had very few logical errors in it that got picked up in simulation, most errors arose from misunderstandings in the hardware and the language, for example not realising some peripherals were active low (which obviously did not appear as a problem in the simulation) and not knowing that a process could only read inputs in its sensitivity list (for a long time, the assumption was that only inputs which we wanted to trigger the process should be in the sensitivity list, not all the inputs we wanted the process to read). This meant that whilst the simulation showed us there was a problem and maybe which module it was in, it provided very little further information most of the time. There were also one or two cases were the system worked as expected in the simulation, but this did not translate to the real board, due to real-world timing errors or similar.

## Conclusion:

During this project, several problems were encountered. The first major obstacle was observing the behaviour of any given program, due to insufficient understanding of how to use test benches for simulations, and the long period of time required to generate and write bitstreams for hardware testing.

Initially it was not understood that signal states could be forcibly modified while a simulation was running, or that a simulation could be stepped forward in time increments, rather than running for a preset time. This was initially solved by using timers to trigger state changes to simulate button presses and such, however this was tedious to implement, and difficult to get the timing right for a simulation running for a fixed duration.

Once it was better understood how to properly utilize testbench simulations, it became much easier to use them to understand the behaviour of all parts of the program and diagnose any incorrect behaviour.

Over the course of this project, several important lessons were learned, and skills developed. The most important was gaining a basic familiarity with VHDL, in particular how to create appropriate architectural models for different uses, and then implement them as entities and connect them with signals and ports as necessary to attain the desired behaviour. Additionally, learning to properly utilize testbench simulations to aid design and testing of a program was also a significant learning outcome from the project. A more minor but still interesting lesson was how to drive a multiple-digit display with just a few signals by displaying only one digit at a time, but alternating between each digit in quick succession to create the illusion that all digits were simultaneously active.

There are some potential improvements could be made to both the program that was produced, for example it could have used a more intuitive indicator of maximum, minimum, and average displays than the single line segments of varying heights as implemented. These were deemed to be a sufficient compromise between readability and complication to implement though.

## References

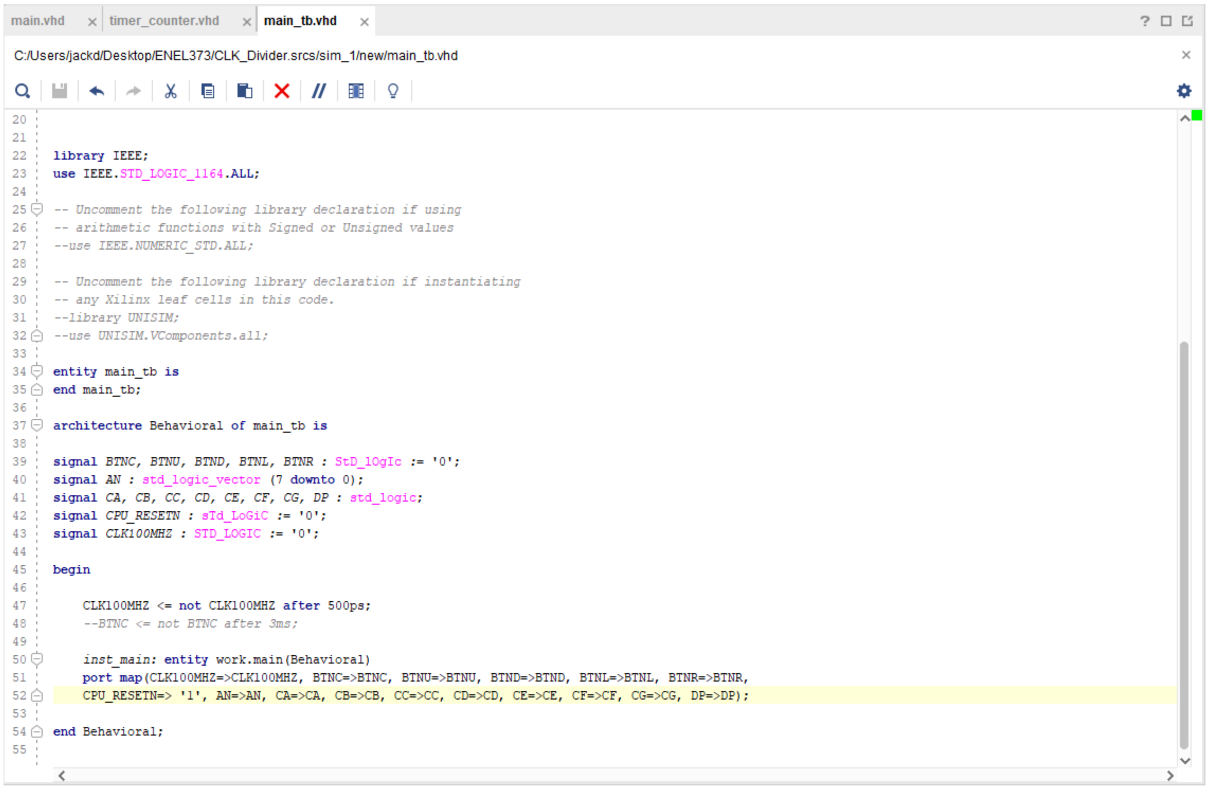
[1] University of Canterbury - ENEL373, "Labs + Reaction Timer Project," 19 2 2024. [Online]. Available: https://learn.canterbury.ac.nz/mod/folder/view.php?id=3524022. [Accessed 19 2 2024].

## Appendices:

*There is no need to include a full code listing in your report as your code will be submitted via eng-git. However, reference to a code snippet in an appendix is quite appropriate. Formal citations to books, web articles, open source VHDL code and other sources should be listed in a reference section that uses the IEEE style and format.*

### Appendix A

This appendix shows the testbench for the main file in figure 2 and specifically the waveform showing how one of the decade counters used for the display works form the main testbench in figure 3 below.



C:/Users/jackd/Desktop/ENEL373/CLK_Divider.srcs/sim_1/new/main_tb.vhd

```vhdl


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity main_tb is
end main_tb;

architecture Behavioral of main_tb is

signal BTNC, BTNU, BTND, BTNL, BTNR : StD_lOgIc := '0';
signal AN : std_logic_vector (7 downto 0);
signal CA, CB, CC, CD, CE, CF, CG, DP : std_logic;
signal CPU_RESETN : sTd_LoGiC := '0';
signal CLK100MHZ : STD_LOGIC := '0';

begin

    CLK100MHZ <= not CLK100MHZ after 500ps;
    --BTNC <= not BTNC after 3ms;

    inst_main: entity work.main(Behavioral)
    port map(CLK100MHZ=>CLK100MHZ, BTNC=>BTNC, BTNU=>BTNU, BTND=>BTND, BTNL=>BTNL, BTNR=>BTNR,
    CPU_RESETN=> '1', AN=>AN, CA=>CA, CB=>CB, CC=>CC, CD=>CD, CE=>CE, CF=>CF, CG=>CG, DP=>DP);

end Behavioral;

```

**Figure 2** Testbench for the main file.

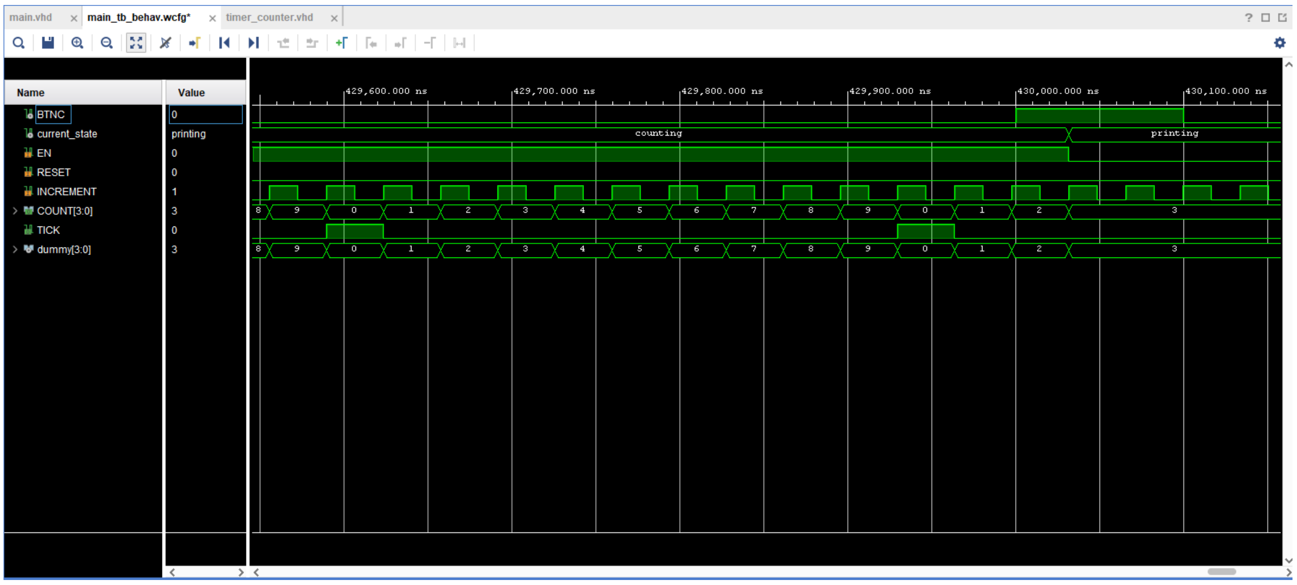
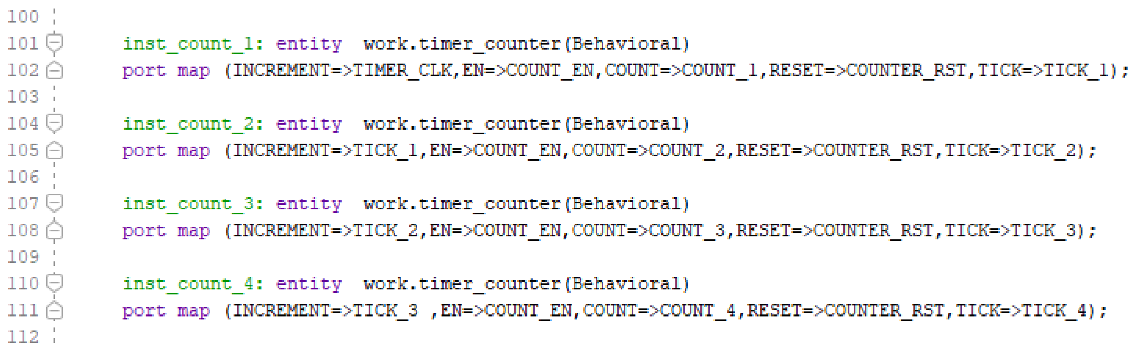


**Figure 3** Waveform of a working decade timer going from counting state to printing state.

## Appendix B

This shows the VHDL code of multiple decade timers being used to get a larger overall timer.



```
    inst_count_1: entity  work.timer_counter(Behavioral)
    port map (INCREMENT=>TIMER_CLK,EN=>COUNT_EN,COUNT=>COUNT_1,RESET=>COUNTER_RST,TICK=>TICK_1);

    inst_count_2: entity  work.timer_counter(Behavioral)
    port map (INCREMENT=>TICK_1,EN=>COUNT_EN,COUNT=>COUNT_2,RESET=>COUNTER_RST,TICK=>TICK_2);

    inst_count_3: entity  work.timer_counter(Behavioral)
    port map (INCREMENT=>TICK_2,EN=>COUNT_EN,COUNT=>COUNT_3,RESET=>COUNTER_RST,TICK=>TICK_3);

    inst_count_4: entity  work.timer_counter(Behavioral)
    port map (INCREMENT=>TICK_3 ,EN=>COUNT_EN,COUNT=>COUNT_4,RESET=>COUNTER_RST,TICK=>TICK_4);
```

**Figure 4** Multiple instances of the same module for a decade timer being used to get a larger count from 0 to 9999.

Code for the decade timer in module file timer_counter.

```
----------------------------------------------------------------------------------
-- Engineer: Jack Edwards, Rob Bass, Brent Smith
--
-- Create Date: 08.03.2024 11:51:25
-- Module Name: timer_counter - Behavioral
-- Project Name: Reaction Timer
-- Description: a decade timer counts from 0 to 9
----------------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

use IEEE.NUMERIC_STD.ALL;
```

```
-- inputs and outputs
entity timer_counter is
   Port ( EN : in STD_LOGIC;
          RESET : in STD_LOGIC;
          INCREMENT : in STD_LOGIC;
          COUNT : out STD_LOGIC_VECTOR (3 downto 0);
          TICK : out STD_LOGIC);
end timer_counter;

architecture Behavioral of timer_counter is
   -- dummy signal for an internal version of the count output
   signal dummy : std_logic_vector (3 downto 0);
begin
   decade : process (RESET, EN, INCREMENT) is
   begin
   -- check for resetting the counter
   if RESET = '1' then
      dummy <= (others => '0');
      TICK <= '0';
   elsif rising_edge(INCREMENT) then
      -- checks timer is enabled then adds 1 and resets TICK to zero
      if EN = '1' then
         dummy <= std_logic_vector(unsigned(dummy) + 1);
         TICK <= '0';
         -- checks if counter is 9 then puts back to zero
         -- then sets the TICK output to 1
         if dummy = "1001" then
            dummy <= "0000";
            TICK <= '1';
         end if;
      end if;
      end if;
   end process decade;
   count <= dummy;
end Behavioral;
```