Final Project: Reaction Timer

Pre-lab: Not graded, but needs to be included into Project Report
Demo Due (50 points): Nov 14 Thursday, 3:00 PM EST
Report Due (50 points): Nov 20 Wednesday, 5:00 PM EST
Total Points: 100

Reminder of grading policy:

1. The overall course grade will be a weighted average of: Labs 60%, Final Project: 30%, Attendance: 10%.
2. As a bonus, you will get 1 extra point for completing the course survey. This will be set up on ELC later and you will get emails about it.

Notes for the final project:

1. You MUST use the one-procedure design method for your Reaction Timer. You CANNOT use the two-procedure design method. Otherwise, you will lose 50% of your Project credit.
2. Consider the button-bouncing problem in your Reaction Timer design and how it will affect your design logic.
3. Every module you design or integrate has to be test-benched. This will be part of demo and graded in your project report.
4. Remember that you must use a High Level State Machine for your design.
5. On average, teams spent ~10 hours to complete the project.
6. TAs and I will help with high level design questions. Your team is responsible for debugging. Please see attached self-checklist for your dubugging reference.

Available Files on ELC (Pre-Designed components):

- Top Level Design Module (Top.v)
- User Constraints File (Top.ucf)
- Clock Divider for LCD Interface (LCDClkDiv.v)
- LCD Display Controller Module (LCDDisplay.v)
- LCD Interface Module (LCDInterface.v)
- 10-bit Binary to 4 Digit BCD Converter (Bin2BCD.v)

Skeleton Files (Components you need to work on):

- Skeleton Verilog Module for Reaction Timer (ReactionTimer.v)
- Skeleton Verilog Module for RandomGen (RandomGen.v)
- Skeleton Verilog Module for ClkDiv (ClkDiv.v)

Lab Overview:

In this final project, you will be designing a **Reaction Timer** that measures a user's reaction time by counting the time elapsed between illuminating a set of LEDs and the pressing of a button by a user.

- *For this implementation you will interface a Pmod CLP 16x2 Character Display LCD with the Basys2 Board. The way Pmod CLP LCD is attached to Basys2 board is found in below figure. The datasheet can be found at this link. (or use eLC for digital copy)*
- *On **Reset**, the Reaction Timer will initially display an introduction message "Reaction Timer" on the connected LCD to the Basys2 board.*
- *When the **Start** button is pressed, the reaction timer will wait for a random length of time between **1 and 3 seconds** while displaying "Wait for LEDs..." on the LCD.*
- *The reaction timer will then illuminate all eight of the individual LEDs of the Basys2 board and **measure the length of time** before the user presses the **Start** button again. The measured reaction time will be displayed on the LCD display in the following format: "0.345s".*
- *If the user did not press the **Start** button within **0.500 seconds (500 milliseconds)** of illuminating the LEDs, the reaction timer will display "Too Slow!" on the LCD.*
- *Alternatively, if the user presses the **Start** button before illuminating the LEDs, the reaction timer will display "No Cheating!" on the LCD.*
- *The Reaction Timer will continue to display the last recorder reaction time, cheat message, or slow message, until the user pushes the **Start** button to perform a new measurement.*
- *Included at the end of this lab is a self-checklist, things you need to consider and test for prior to asking for help from the TAs/Instructor. After you have exhaustively checked the self-checklist, please refer your inquiry to the TA/Instructor. This checklist is meant to help you debug your own code.*

The following Figure 1 provide a top level view of the various components of the Reaction Timer you will be designing and provide an overview of the connections that will be used to implement your Reaction Timer on the Basys2 board. The shaded components are **predesigned components** provided to you that you must interface with to implement the Reaction Timer design (available to you on ELC final project folder). The top level User Constraint File (Top.ucf) is also available to you on ELC. Please check carefully in the .ucf file the mapping between inputs/outputs and physical addresses.

Figure 2 shows the connection between Basys2 board and the Pmod CLP LCD module. Please note that this is the connection you need to follow in order for the provided .ucf file to work.

Figure 3 shows the pins of the Pmod CLP LCD module.

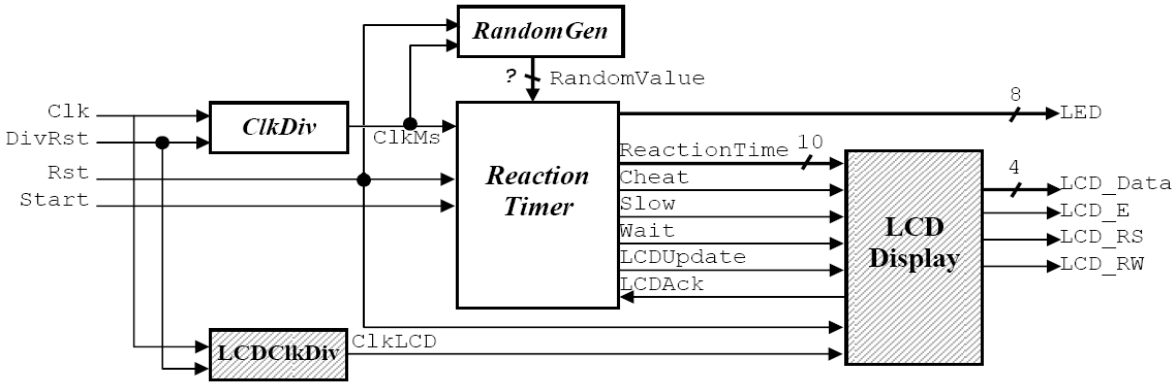


Figure 1. Block Diagram of Reaction Timer (Shaded components are predesigned)

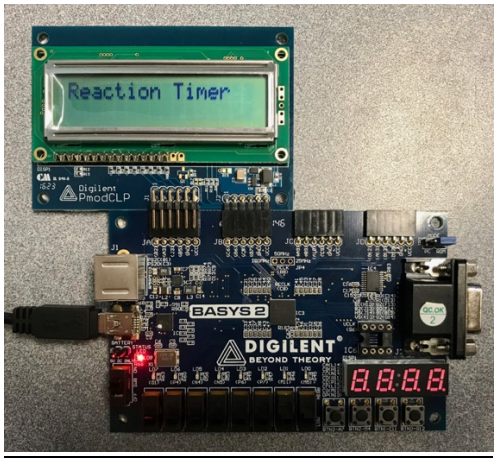


Figure 2. Please connect Basys2 board to Pmod CLP LCD module following above figure. The provided .ucf file necessitates this particular connection.

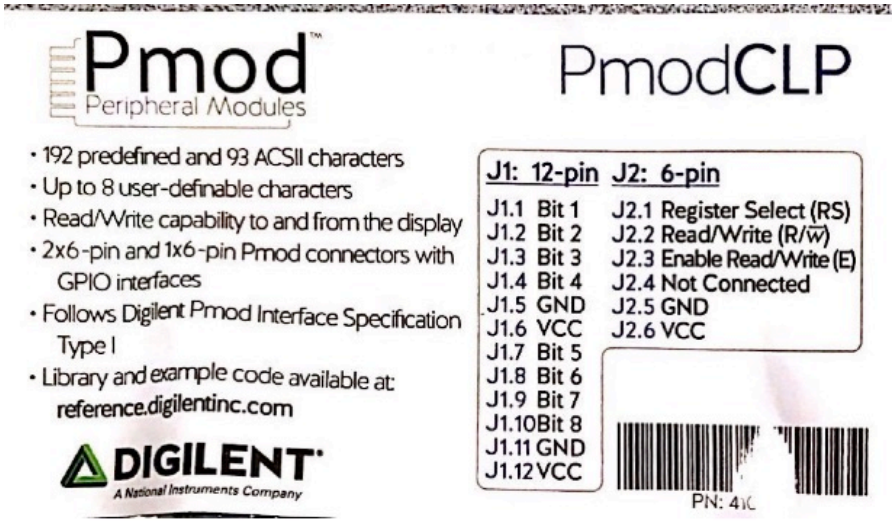


Figure 3. Pmod CLP LCD module's pins.

Design of Reaction Timer:

The Reaction Timer component provides the main functionality of the overall design and will interface with the clock divider (ClkDiv), the random number generator (RandomGen), and the LCD display (LCDDisplay). In this project, you will create an **RTL design (with one-procedure Verilog method)** of the above specified reaction timer as high-level state machine (HLSM). The Reaction Timer component has five inputs, **ClkMS, Rst, Start, LCDAck, and RandomValue**, and six outputs, **LED, ReactionTime, Cheat, Slow, Wait, and LCDUpdate**.

The **ClkMS** input is a 1kHz clock signal generated by the ClkDiv component. While the Reaction Timer's HLSM could directly operate using the 50MHz clock provided by Basys2 Board board, a 1KHz clock signal provide a simplified method for counting the elapsed time in milliseconds – 1 clock cycle of a 1 kHz clock is equal to 1 millisecond. You will need to design the ClkDiv component that will generate a 1 kHz clock output, ClkMS, from the 50 MHz clock input provided by Basys2 board.

The **RandomValue** input is an N-bit number output from the RandomGen component that provides a random value between 1 and 3 seconds used to delay the illumination of the LEDs for a random length of time. As there are many ways to design the RandomGen component, the number of bits used to represent the RandomValue signal is not defined. Instead, in designing the RandomGen component, you will need to determine the number of bits needed for this value. The RandomGen component doesn't have to implement a true random number generator, or pseudo-random number generator. Instead, the RandomValue should appear to be random when read by the Reaction Timer at the appropriate time. Please note that the built-in Verilog function **$random** is not synthesizable and should not be used.

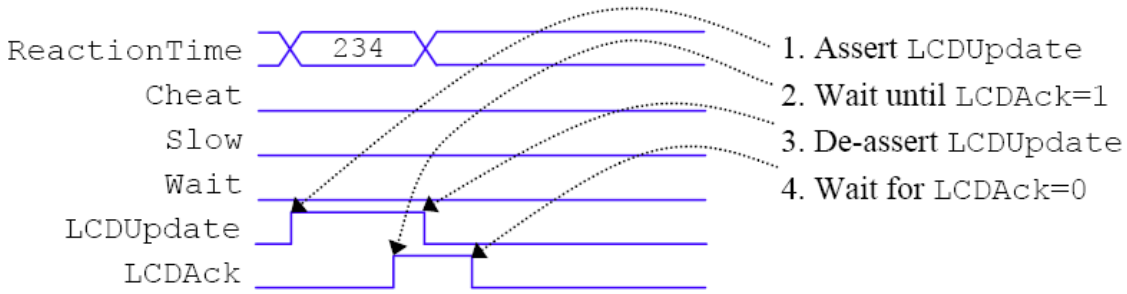
The **LED** output is an 8-bit output that will connect to the eight individual LEDs available on the Basys2 board. The Reaction Timer should illuminate all eight LEDs to indicate that the user should press the Start button as soon as possible. In all other circumstances, the LEDs should not be illuminated.

The input **LCDAck** and outputs **ReactionTime, Cheat, Slow, Wait, and LCDUpdate** are used to interface with the provided **LCDDisplay** component. The **ReactionTime** output is a 10-bit output that corresponds to the measured time in milliseconds of a successful reaction time measurement. The **Cheat** output is a one-bit output indicating the user pressed the Start button before the LEDs were illuminated and the "No Cheating!" message should be displayed on the LCD. The **Slow** output is a one-bit output indicting the user did not press the Start button within 0.500 second after illuminating the LEDs and the "Too Slow!" message should be displayed on the LCD. The **Wait** output is a one bit output indicating that the "Wait for LEDs..." message should be displayed on the LCD.

Finally, the **LCDUpdate** output and **LCDAck** input are used for communicating with the pre designed **LCDDisplay** component to specify when and how the LCD display should be updated. Because the **LCDDisplay** and Reaction Timer components operate using two different clock signals, in order to communicate between the two components, your design will have to implement a simple handshaking scheme. **As illustrated in the following timing diagram, to update the LCD, the Reaction Timer must first assert the LCDUpdate output and continue to assert this output until the LCDAck input is asserted**

by the LCDDisplay component. Upon detecting that the LCDAck input is high, the Reaction Timer must de-assert the LCDUpdate signal and wait for the LCDAck input to be de-asserted as well. This handshaking sequence is required for all updates to the LCDDisplay.

For example, to display a measured reaction time on the LCD, the Reaction Timer should assign the measured reaction time in milliseconds to the ReactionTime output, assign 0 to Cheat, Slow, and Wait, assert the LCDUpdate output, and complete the handshaking as described above. Note that the values assigned to ReactionTime, Cheat, Slow, and Wait must remain constant as long as LCDUpdate is asserted. As another example, to display the wait message on the LCD, the Reaction Timer should assign 1 to the Wait, assign 0 to Cheat and Slow, assert the LCDUpdate output, and complete the handshaking as described above.



The provided top-level module (Top.v) integrates the various components together in a structural manner to implement the overall reaction timer design. In designing the Reaction Timer, RandomGen, and ClkDiv components, be sure to use consistent module names to ensure your components will integrate effortlessly. Skeleton Verilog modules are also provided for the Reaction Timer (ReactionTimer.v), RandomGen (RandomGen.v), and ClkDiv (ClkDiv.v) components.

Pre-lab (Not graded, but needs to be included in report):

***It is IMPERATIVE for you to complete the pre-lab first, before going into design and implementation ***

1. Behaviorally design the **ReactionTimer** component as a high-level state machine.
    - Create a logical sequence of events corresponding to the Reaction Timer.
    - Describe the Reaction Timer's behavior as a high-level state machine. The state machine consists of states and transitions. You need to **draw the state diagram**, **specify the initial state**, **input/output/registers (multiple bit or single bit)**, **state**, **transition**, **state action**, **arithmetic operation**, etc. You need to decide the number of data registers needed for these inputs/outputs and temporary storage elements, including their bits. This diagram will be used to grade your project report, and you need to be able to explain the design thoughts behind the diagram.

2. Behaviorally design the RandomGen component. (*Hint: The RandomGen component need not explicitly generate subsequent random numbers, but rather should appear to be random with respect to the user's interactions with your design.)*

3. Design the clock divider, ClkDiv.

Demo (50 points):

1. **(5 points)** Testbench and simulation waveforms demonstrating correct functionality of the **RandomGen** component. Convince the TA with waveforms that you are indeed generating random numbers in with correct bit length. The number should appear to be random.

2. **(10 points)** Testbench and simulation waveforms demonstrating correct functionality of the **Reaction Timer** component. You need to convince the TA using waveforms that this component can perform all necessary tasks (**reset, measure reaction time, detect cheating, detect slow response, start another measurement after the previous one**). The waveforms need to provide convincing evidence that the component is **measuring the correct length of time of reaction**, **slow response is indeed a response after 0.5 second**).

3. **(10 points)** Testbench and simulation waveforms demonstrating correct functionality of the connected **Reaction Timer** and **RandomGen** components. Create a testbench that connects the Reaction Timer and RandomGen components together. Your waveform should convince the TA that all tasks of reaction timer (**this time, including the wait of random length of time**) are accomplished. The waveform should provide convincing evidence that your reaction timer is really taking the random number generated and using it for waiting. (*Note: As you are not yet interfacing with the LCDDisplay component, you will need to control the LCDAck signal from within your testbench to provide and acknowledge in response to the LCDUpdate signal.)*

4. **(5 points)** Behavioral design of ClkDiv circuit generating 1 kHz clock output. Testbench and simulation waveforms demonstrating correct functionality of the ClkDiv component for one clock period (1 millisecond).

5. **(20 points)** Synthesis and implementation of your overall design to the Basys2 board demonstrating correct reaction timer functionality (**reset, measure reaction time, detect cheating/slow response, wait random length of time, start another measurement after the previous one**). It is difficult to benchmark the timer during this demo. The TA will rely on your testbenches and waveforms to verify whether the numbers are correct.

6. **After completing the demo, TA will check the components of the FPGA board package, including a working Basys2 board, power cable, USB cable, and a Pmod CLP LCD module. The team needs to have the complete package and return it to the cabinet in Room 1214, before the demo credit is awarded.**

Self Debugging Checklist:

Design logic parts:

- Is your design approach logical? Are you choosing the right approach for modules? For example, Boolean/truth tables for combinational logic problems, FSM for sequential problems, and HLSM for processors?
- If for FSM/HLSM, are you starting with an appropriate list of sequence of events? Are you listing and accounting for all the inputs/outputs/data registers?
- If for FSM/HLSM, are you using sufficient bits for state registers?

Verilog:

- Are you using one-procedure or two-procedure for HLSM? This depends on the lab instruction.
- Are you using behavioral approach or structure approach? Behavioral approach is for functional modules, and structure approach is for top level/sub-top level modules.
- Did you declare inputs/outputs and set them to proper types of variables?
- Is your sensitivity list appropriate?
- Are you using synchronous design? It means FSM/HLSM transitions based on rising edges of clock.
- Did you have an appropriate list of sequence of events?
- Did you have a complete state diagram based on the sequence of events?
- Did you register all multiple-bit outputs?
- Did you create data registers for all temporary storage elements?
- Did you set all state actions (outputs) in each state?
- Is your FSM/HLSM safe? What if it gets into an un-declared state?
- Are you using the right assignment, non-blocking assignment or blocking assignment? Block assignment needs to be used when there is arithmetic operations.

Simulation:

- Remember, test bench files are for simulation only and functional module files are for both simulation and implementation. Always set a file based on its type at the time of its creation.
- If for FSM/HLSM, did you properly reset your FSM/HLSM at the very beginning? Was your reset signal long enough to cover at least one clock cycle. It is better to cover more than one clock cycle. Was your clock signal a true clock signal or a slowed signal?
- If your waveform looks wrong, did you check your FSM/HLSM state and did they transition correctly? Do you know how to display states on your waveform? Do you know how to display any variable that is not an input/output on your waveform? If not, please ask your TAs.
- Did everything work (truthfully, without a doubt) before your start implementing?
- Are you test benching the right file? If so, have you initialized the registers by implementing a reset? Is your ClkDiv displaying a frequency too large for the waveform window? Make sure to walk through your code line by line and verifying the appropriate behavior in the wave.

Implementation:

- Did you set the board to be the right board? Refer to lab 1 for specs.
- Did you set the top module to be the one that was implemented? Again, do not try to implement a test bench file.
- Did you have a user constraint file (.ucf)? Is it correctly tied to the top module? Try recreating your .ucf file and pasting the appropriate mappings.
- Did you map all inputs/outputs to the correct physical addresses on the board?

Notes:

1. **You must use a USB drive for all labs. You cannot use network drive or Z drive. Usage of network drive and Z drive have created weird problems in the past and TAs won't be able to help you.**
2. **Lab report from each team will need to follow the specific format and contents, the template of lab reports are in ELC named as "Lab_Report_Template".**
3. **Grading Policy:**
    a. Only one copy of the lab report from each team is needed. The grade for the report will apply to both team members, provided that each team member contributes roughly equally (50/50 – 40/60) to the lab. In the case where one team member contributes disproportionally lower to the lab, the instructor may reduce the grade of this member. Late submission policies are listed below.
    b. For pre-lab: late submission after due time will receive partial credits (70% of credit for 0-1 hour late submission, no credit for more than 1 hour late submission).
    c. For demo: late demo after due time will partial credits (70% of credit for 0-1 hour late submission, no credit for more than 1 hour late submission).
    d. For lab report: late submission after due time will receive partial credits (70% of credit for 0-24 hour late submission, no credit for more than 24 hours late submission).
    e. Plagiarism: If plagiarism is found in reports, both teams will get ZERO points immediately, and be reported to University.