

# Simulating a Simple Design

2021.1

## Abstract

This lab emphasizes the flexibility of processes within a testbench. It should take approximately 45 minutes.

## Objectives

After completing this lab, you will be able to:

- Demonstrate the use of processes to generate clock and reset stimulus
- Use the Vivado® simulator to navigate the output waveform

## Introduction

During this lab, you will create processes that generate a 50/50 duty cycle clock and a reset that always deasserts after 25 clock pulses regardless of the clock period. A data generation module is provided and must be instantiated within the testbench.

You will be guided through some of the more advanced features of the Vivado simulator.

## Understanding the Lab Environment

The labs and demos provided in this course are designed to run on a Linux platform. Many of the labs and demos can be successfully executed in the Windows environment or a native Linux environment as well.

The instructions found in this lab are expressed using the Linux notation. This includes the forward slash ('/') as the hierarchy separator instead of the Windows backslash ('\'). Students who want to run the labs directly under Windows must use the correct hierarchy separator.

Customizable environment variables enable you to tailor your environment for specific machine configurations. The only environment variable (shown below) used in the customer training environment (CustEd\_VM) points to the training directory where all the lab files are located. This reduces the amount of typing you need to do when entering directory paths.

This environment variable can be customized according to your specific location and can be set for Linux systems in the `/etc/profile` file and for Windows systems by entering "env" from the search bar.

The following is the environment variable used in the customer training VM:

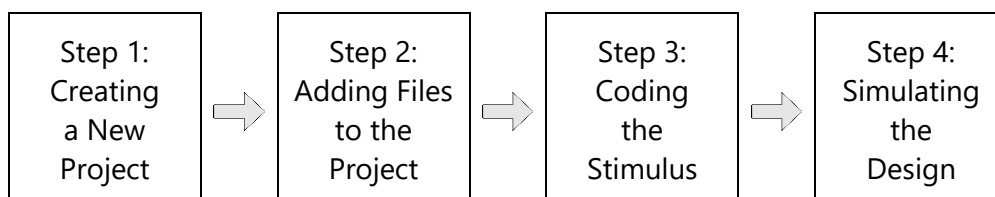
Environment Variable Name	Description
<code>\$TRAINING_PATH</code>	<p>Points to the space allocated for students to work through the labs. This directory includes prebuilt images and starting points for the labs and demos.</p> <p>The customer training VM sets <code>\$TRAINING_PATH</code> to the <code>/home/xilinx/training</code> directory.</p> <p>Typically, Windows users will install the training directory under C: to keep the path names as short as possible.</p>

**Note:** Environment variables are not supported from the Vitis IDE GUI. When using this tool, you must manually replace `$TRAINING_PATH` with the value of the variable, which in the customer training virtual machine, is `/home/xilinx/training`. Other tools, such as the Vivado Design Suite, will properly expand the environment variable.

Additional note about environments: Both the Vivado Design Suite and Vitis platform offer a Tcl environment. The contents of this environment are NOT preserved with the project. When the tools launch, they start with a pristine Tcl environment with none of the procs or variables remaining from a previous launch of the tools.

This means that if you sourced a Tcl script or manually set any Tcl variables and you closed the tool, when you reopen the tool (and perhaps even reopen the last project), you will need to source the Tcl script again and set any variables that the lab requires.

## General Flow



---

## Creating a New Project

## Step 1

---

### 1-1. Launch the Vivado Design Suite.

If you do not recall how to perform this task, refer to the "Launching the Vivado Design Suite" section under Vivado Design Suite Operations in the *Lab Reference Guide*.

### 1-2. Create a new Vivado Design Suite project named *basicTestbench* and locate it in the following directory.

Browse to the `$TRAINING_PATH/basicTestbench/lab/KCU105` directory.

Target the project for the Kintex- UltraScale KCU105 Evaluation Platform evaluation board.

If you do not recall how to perform this task, refer to the "Creating a Blank Vivado Design Suite Project" section in the *Lab Reference Guide*.

From the settings in the Flow Navigator, change the target language to VHDL.

In this lab, you will not generate the bitstream and verify the design on hardware; it does not require the KCU105 board for you to download the bitstream, just the project you will be targeting to the KCU105 board.

## Adding Files to the Project

## Step 2

The `uart_led` design is a sub-set of the larger `wave_gen` design which is the common design used in the many of Xilinx's courses.

The source files for the `uart_led` have been preloaded into the working directory and you will be using these design files to perform simulation.

### 2-1. Add the following design files into the project.

- **LED\_manager** – multiplexer to select functionality
- **meta\_harden** – anti-meta-stable circuit
- **register8.vhd** – 8-bit registering flip-flop module
- **reset\_bridge** – for safely applying reset
- **uart\_baud\_gen** – baud rate generator for oversampling serial data
- **uart\_driver\_tb** – creates stimulus for the testbench
- **uart\_led.vhd** – top level of the design
- **uart\_led\_simple\_tb** – simplified testbench
- **uart\_rx.vhd** – UART receiver
- **uart\_rx\_ctl** – UART state machine

2-1-1. Click **Add Sources** under Project Manager in the Flow Navigator.

The Add Sources Wizard opens.

2-1-2. Select **Add or create design sources**.

2-1-3. Click **Next**.

2-1-4. Click the **Plus (+)** icon and select **Add Files**.

2-1-5. Press the **<Ctrl>** key and select all the files listed above from the location  
`$TRAINING_PATH/basicTestbench/support`.

2-1-6. Click **OK**.

2-1-7. Click **Finish** to confirm adding the selected files to the project with the default associations and libraries.

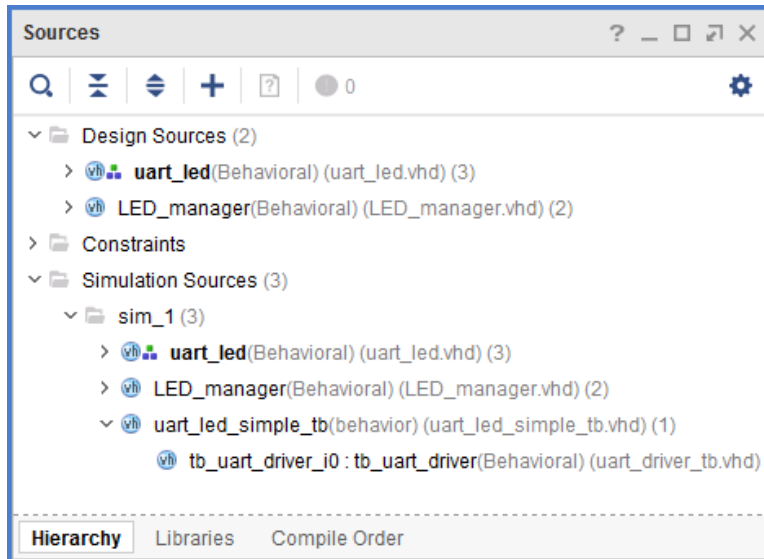
The files that are added can now be seen in the Sources view. The `uart_led_simple_tb` and `tb_uart_driver` files are simulation only files and should be moved to the simulation only category.

## 2-2. Associate the testbench files to simulation from the Sources view.

### 2-2-1. Right-click **uart\_led\_simple\_tb** under Design Sources and select **Move to Simulation Sources**.

The file is now moved under Simulation Sources in the Sources view.

### 2-2-2. Repeat this step for **tb\_uart\_driver** to be moved under Simulation sources.



**Figure 6-1: Sources Window**

You will also see some of the other modules that are instantiated in `uart_led_simple_tb` are also seen in the simulation hierarchy.

## 2-3. Open the top file of the design, examine the contents, and complete the code to make necessary connections as shown in the figure below.

**uart\_led.vhd** is the top-level source file for the **uart\_led** project and will contain the completed code for synthesis. All that remains is to properly connect **LED\_manager**.

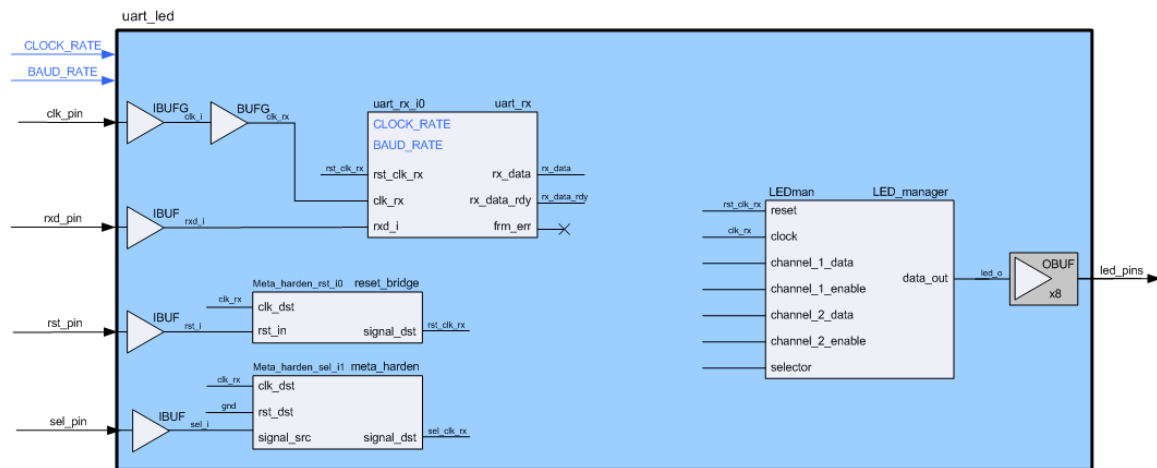


Figure 6-2: Block Diagram of **uart\_led** design

2-3-1. Double-click **uart\_led** in the Hierarchy view to open the source file.

2-3-2. Uncomment the **LED\_manager** instantiation at lines 191 to 200.

## 2-4. Output of the UART receiver must be connected to the LED\_manager.

**Make the appropriate signal connections to the LED\_manager instantiation in **uart\_led** and save the file.**

2-4-1. Connect **rx\_data** to Channel\_1\_data and **rx\_data\_rdy** to Channel\_1\_enable.

Connect the meta-hardened selection line **sel\_i** from **meta\_harden\_sel\_i1** to the selector input on the **LED\_manager**.

The clock input to the **LED\_manager** is driven by **clk\_rx**.

Not surprisingly, the reset of **LED\_manager** is connected to **rst\_clk\_rx**.

The **data\_out** from the **LED\_manager** should tie to the inputs of the **OBUFs** using **LED\_o**. This leaves the inputs to **Channel\_2** open.

2-4-2. Tie these signals to ground using the constant named **gnd**.

Note that the data is the full 8 bits, but the enable is only 1 bit. Use a slice of **gnd** (i.e., **gnd(0)**) for the enable.

2-4-3. Select **File > Text Editor > Save File**.

**2-5. Synthesize the design with the default options.**

- 2-5-1. Click **Run Synthesis** in the Flow Navigator and click **OK**.
- 2-5-2. Correct any errors if found and resynthesize.
- 2-5-3. Click **Cancel** if the Synthesis Completed dialog box appears.

**Coding the Stimulus****Step 3**

In this step, you will step through the creation of the stimulus for `uart_led` design. The data generation module, which uses a number of techniques that will be introduced later in this class, has been provided for you.

You will first create a clock which produces a periodic 50% duty cycle square wave running at 100 MHz. You will then build a mechanism that holds reset asserted until 100 clock cycles have passed. Next, you will build a selector to switch between the grounded Channel 2 and the received data. Finally, you will connect all the blocks of code together and check syntax.

**3-1. Using the coding style of your choice, create a block of code in the `uart_led_simple_tb.vhd` file to create a 100-MHz clock oscillator with a 50% duty cycle.**

**Although a suggested code solution can be found in the Answers section, you should make every effort to code this yourself.**

- 3-1-1. Expand **Simulation Sources** in the Sources view and double-click `uart_led_simple_tb.vhd` to open the testbench source file.
- 3-1-2. Write a process or concurrent statement that generates a 100-MHz clock below the line of code which reads "-- generate the clk\_rx signal" on or near line 144.

**Hint:** Use the `CLK_RX_PERIOD` constant to compute the half period.

**3-2. The next block of code that needs to be written is that of the reset.**

**Code the reset so that it is active high for the first 100 clock periods of the simulation then drops low.**

**Although a suggested code solution can be found in the Answers section, you should make every effort to code this yourself.**

- 3-2-1. Write a process or concurrent statement that generates an active high reset for 100 clock pulses below the line of code which reads "-- generate the reset signal".

**Hint:** Use the `CLK_RX_PERIOD` constant to delay the deassertion of reset.

- 3-3. The next task is to create a selector stimulus. Because the serial data being pumped out of `tb_uart_driver` is at 115200 baud, it can safely be assumed that one "frame" of data will be sent in approximately 100  $\mu$ s. There are 45 characters to send.**

**Therefore, select Channel 1 and hold for 4.5 ms, then switch to Channel 2. If everything works, this will allow the message to be presented, then zeros after 4.5 ms.**

**Although a suggested code solution can be found in the Answers section, you should make every effort to code this yourself.**

- 3-3-1.** Write a process or concurrent statement that holds select low for 4.5 ms and then switches to Channel 2 (select high) below the line of code which reads "-- generate the select signal".

- 3-4. Because this testbench was modified from an earlier design, verify that the connections to `tb_uart_driver` are correct.**

**Although a suggested code solution can be found in the Answers section, you should make every effort to code this yourself.**

- 3-4-1.** Locate `tb_uart_driver` and verify that all the input pins are being driven. If they are not, connect them to the proper sources.

- 3-5. The last portion of preparing the testbench for use is to connect the UUT to the various stimuli.**

**Connect the generated stimulus from the clock, reset, and data generator to the UUT.**

**Although a suggested code solution can be found in the Answers section, you should make every effort to code this yourself.**

- 3-5-1.** Uncomment the lines after the comment "-- instantiate the unit under test" near to the end of the instantiation.
- 3-5-2.** Connect the signals to the appropriate stimulus or sink.
- 3-5-3.** Save the file.



## Simulating the Design

## Step 4

In the final portion of this lab, you will run the simulation and view the waveform.

### 4-1. Set the default run time for the Vivado simulator to 4.5 ms and launch the Vivado simulator.

- 4-1-1. Set the **uart\_led\_simple\_tb.vhd** as top level file by right clicking and selecting **Set as Top**.
- 4-1-2. Select **Settings** in the Flow Navigator.
- 4-1-3. Select the **Simulation** view in the Project Settings dialog box and select the **Simulation** tab.
- 4-1-4. Change the Simulation Run Time default value from 1000 ns to **4.5 ms**.

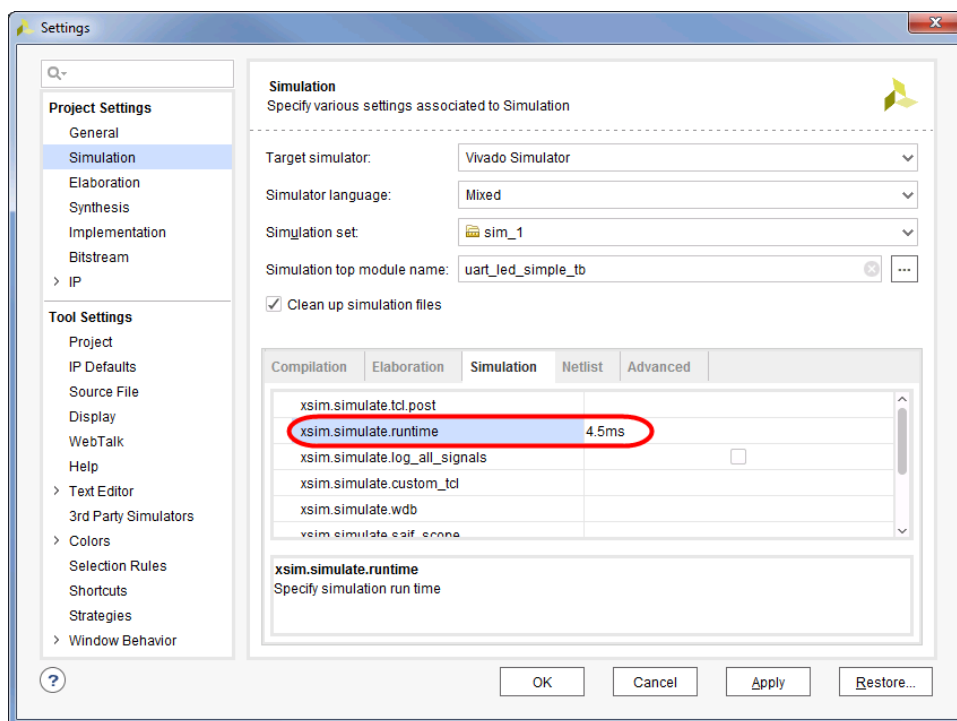


Figure 6-3: Setting New Default Simulation Run Time

- 4-1-5. Click **OK**.
- 4-1-6. Select **Simulation > Run Simulation** in the Flow Navigator.
- 4-1-7. Select **Run Behavioral Simulation**.

#### 4-2. (Optional): Remove the unused signals so that only the following signals are present:

- **clk\_pin**
- **rst\_pin**
- **sel\_pin**
- **led\_pins[7:0]**
- **data\_to\_send[7:0]**
- **data\_sent**
- **rx\_clk\_rx**

4-2-1. Select the signals to remove and press the **<Delete>** key.

#### 4-3. In the simulation waveform, zoom to full screen and zoom in on the first value of led\_pins.

4-3-1. Click the **Zoom Fit** (🔍) icon to zoom to full screen.

4-3-2. Highlight the **led\_pins[7:0]** signal by clicking it once. The waveform will be emphasized.

4-3-3. Right-click **led\_pins[7:0]** and **data\_to send[7:0]** value column and select **Radix > ASCII**.

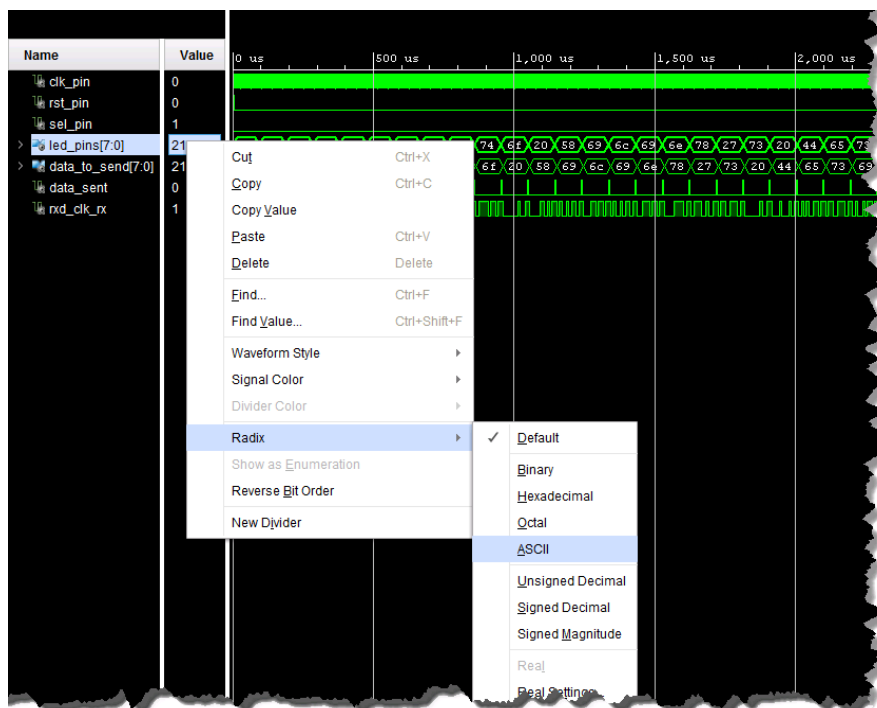


Figure 6-4: Changing the Radix of the Signal

4-3-4. Right-click **led\_pins[7:0]** and select **Signal Color**.

4-3-5. Select **Blue** from the provided palette.

## Question 1

What are the advantages of changing a signal's radix and color?

---



---



---

**4-3-6.** Click the **Go To Time 0** (🕒) icon in the horizontal toolbar of the waveform viewer to move the cursor to time zero.

**4-3-7.** Highlight the **led\_pins[7:0]** signal again by clicking it.

**4-3-8.** Click the **Next Transition** (➡) icon to advance to the next transition of **led\_pins[7:0]**.  
The simulator considers time 0 as the first transition.

**4-3-9.** Click the **Next Transition** (➡) icon once more to get to the transition near 95.385 us.

**4-3-10.** Click the **Zoom In** (🔍) icon to zoom in. Zoom to the point where the values are clearly seen and that both the preceding value of **led\_pins** is also visible.

You may need to pan to get this point visible.

## 4-4. Zoom out to read the message.

**4-4-1.** Click the **Zoom Out** (🔍) icon several times so that the message is clearly seen.

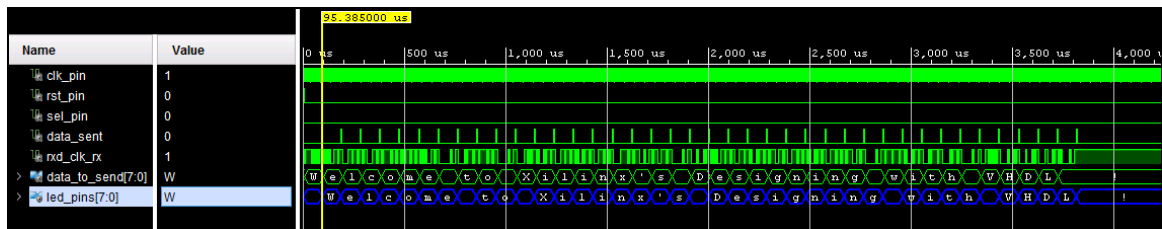


Figure 6-5: Complete Message in the Waveform

## Question 2

Does the received message match the sent message? How can you tell?

---



---




---


**4-5. Simulate for an additional one millisecond and observe the sel\_pin changing to a one and the output of led\_pins at this point.**

**4-5-1.** Enter **1 ms** in the duration field.



**Figure 6-6: Vivado Simulator Control Toolbar**

**4-5-2.** Click the  icon in the horizontal tool bar to run the simulation for 1 ms.

**4-5-3.** Select **sel\_pin** from the Signal Name list and click the **Next Transition** icon (.

You may need to pan or zoom out to see the transition.

This should take you to the point where sel\_pin transitions from a zero to a one.

**4-5-4.** Pan and zoom as necessary to view the output of the LED\_pins after sel\_pin transitions high.

### Question 3

What is the value of LED\_pins when sel\_pin is asserted?

**Hint:** You may have to change the radix of led\_pins[7:0] to binary.

---

---

---

**4-6. Close the simulation.**

**4-7. Close the Vivado Design Suite.**

Some systems (particularly VMs) may be memory constrained. Removing the workspace frees a portion of the disk space, allowing other labs to be performed.

You can delete the directory containing the lab you just ran by using the graphical interface or the command line interface. You can choose either mechanism. Both processes will recursively delete all the files in the `$TRAINING_PATH/basicTestbench` directory.

#### 4-8. [Optional] [Only for local VMs—not for CloudShare] Clean up the file system.

Using the GUI:

**4-8-1.** Using the graphical browser (Windows: press the <**Windows**> key + <**E**>; Linux: press <**Ctrl** + **N**>), navigate to `$TRAINING_PATH/basicTestbench`.

**4-8-2.** Select **basicTestbench**.

**4-8-3.** Press <**Delete**>.

-- OR --

Using the command line:

**4-8-4.** Open a terminal window (Windows: press the <**Windows**> key + <**R**>, then enter **cmd**; Linux: press <**Ctrl** + **Alt** + **T**>).

**4-8-5.** Enter the following command to delete the contents of the workspace:

**[Windows users]:** `rd /s /q $TRAINING_PATH/basicTestbench`

**[Linux users]:** `rm -rf $TRAINING_PATH/basicTestbench`

## Summary

This lab walked you through the creation of stimulus for the reset, clock, and sel\_pin signals using both concurrent statements and processes.

You also learned more of the basic navigational skills and display features of the Vivado simulator.

## Answers

1. What are the advantages of changing a signal's radix and color?

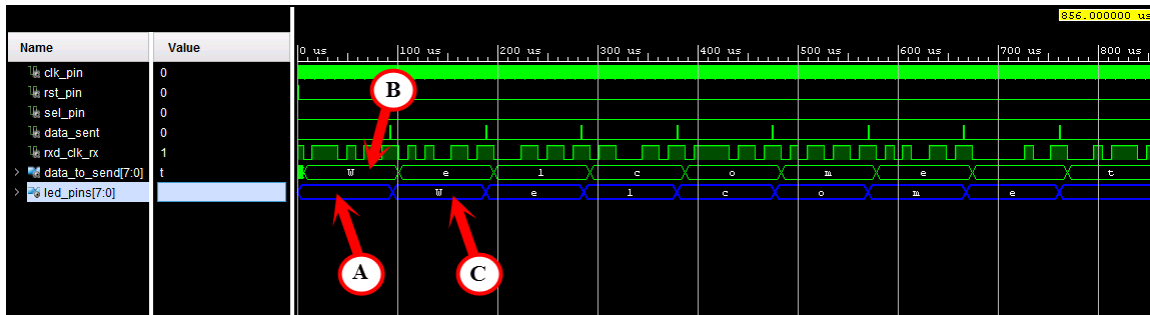
Changing a signal's radix enables you to view the waveform in its native, or most meaningful, representation. Because binary is one of the longest reorientations, virtually all other displays yield shorter labels, enabling you to see more of the waveform at one time.

Changing signal colors allows you to highlight or apply special meaning to signals. For example, signals directly connected to a particular module might be shaded red and another module, blue.

2. Does the received message match the sent message? How can you tell?

There are two ways of seeing that the message was sent.

Each data\_to\_send is shown during the serial transmission and is complete when the data\_sent signal asserts for one clock period. Each transmitted character can be manually compared to each received character, which is presented slightly later on led\_pins.



**Figure 6-7: Manually Verifying the Character Sequence**

'A' – Prior to the transmission of the first character, the output of the UUT is undefined and is shown (in ASCII mode) as a null character.

'B' – The data generator produces the first character and begins transmitting it to the UUT.

'C' – After several clock cycles, the received character is presented on led\_pins. Because this character matches the one sent (indicated by 'B'), you can consider this a successful transmission of this character.

The second message requires you to open the `uart_led_simple_tb.vhd` file and look at the message passed into the `tb_uart_driver`:

```
tb_uart_driver_i0: tb_uart_driver
  generic map (BIT_PERIOD => BIT_PERIOD, MESSAGE => "Welcome to Xilinx's
Designing with VHDL class!")
  port map ( rst_clk_rx      => rst_pin,
             data_to_send    => data_to_send,
             data_sent        => data_sent,
             data_out_serial  => rxd_clk_rx
             );
```

From inspection, the message that appears in the waveform matches that specified in the generic.

- What is the value of `LED_pins` when `sel_pin` is asserted?

The value changes from '!' to 0 several clocks after `sel_pin` is asserted. Note that the radix was switched back to binary to avoid any ambiguity in the display of the ASCII character. Non-printing ASCII characters are represented as ' '.

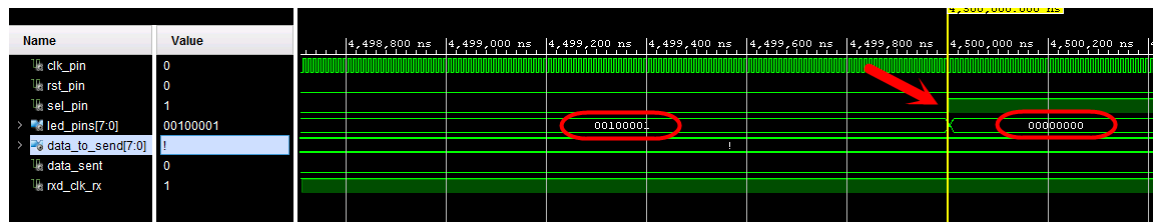


Figure 6-8: Waveform at Assertion of `sel_pin`

## Coding the Stimulus

### Procedure or concurrent statement that generates a 100-MHz clock:

Concurrent (RTL style):

```
clk_pin <= not clk_pin after CLK_RX_PERIOD / 2;
```

OR

Process (behavioral style):

```
makeClk_rx: process
begin
    clk_pin <= '0';
    wait for CLK_RX_PERIOD / 2;
    clk_pin <= '1';
    wait for CLK_RX_PERIOD * 0.5;
end process makeClk_rx;
```

### Process or concurrent statement that generates an active high reset for 100 clock pulses:

Concurrent (RTL style):

```
rst_pin <= '1', '0' after CLK_RX_PERIOD * 100;
```

OR

Process (behavioral style):

```
make_reset: process
begin
    rst_pin <= '1';
    wait for CLK_RX_PERIOD * 100;
    rst_pin <= '0';
    wait;
end process make_reset;
```

-- assert reset



**Process or concurrent statement that holds select low for 4.5 ms, then switches to Channel 2 (select high):**

Concurrent (RTL style):

```
sel_pin <= '0', '1' after 4.5 ms;
```

-OR

Process (behavioral style):

```
make_sel: process
```

```
begin
```

```
    sel_pin <= '0';
```

```
    wait for 4.5 ms;
```

```
    sel_pin <= '1';
```

```
    wait;
```

```
end process make_sel;
```

**Verifying the connections to the tb\_uart\_driver:**

-- Instantiate the data generator - uses the predefined character sequence and transmits bit-by-bit

```
tb_uart_driver_i0: tb_uart_driver
```

```
    generic map (BIT_PERIOD => BIT_PERIOD, MESSAGE => "Welcome to Xilinx's  
Designing with VHDL!")
```

```
    port map ( rst_clk_rx      => rst_pin,  
               data_to_send   => data_to_send,  
               data_sent      => data_sent,  
               data_out_serial => rxd_clk_rx  
            );
```

**Connecting the UUT to signal sources and sinks:**

```
UUT: uart_led
```

```
generic map (CLOCK_RATE => CLOCK_RATE,      -- generics defined in entity  
             BAUD_RATE  => BAUD_RATE  
            )
```

```
port map ( clk_pin    => clk_pin,  
           rst_pin    => rst_pin,  
           rxd_pin    => rxd_clk_rx,  
           sel_pin    => sel_pin,  
           led_pins   => led_pins  
        );
```