EGR 270
Fundamentals of Computer Engineering
Filename: Tutorial 2 – Aldec Active HDL and Xilinx Vivado.docx

Tutorial 2: Sequential Logic Circuits using Aldec Active-HDL and Xilinx Vivado

This tutorial will guide you through specifying a design for a 3-bit up/down counter using Aldec Active-HDL. The software includes a State Diagram wizard that allows you to draw a state diagram, specify paths and conditions, etc. This tutorial assumes that you have a basic familiarization with Aldec Active-HDL. If not, you may want to review **"Tutorial 1 - Combinational Logic Circuits using Aldec Active-HDL and Xilinx Vivado".**

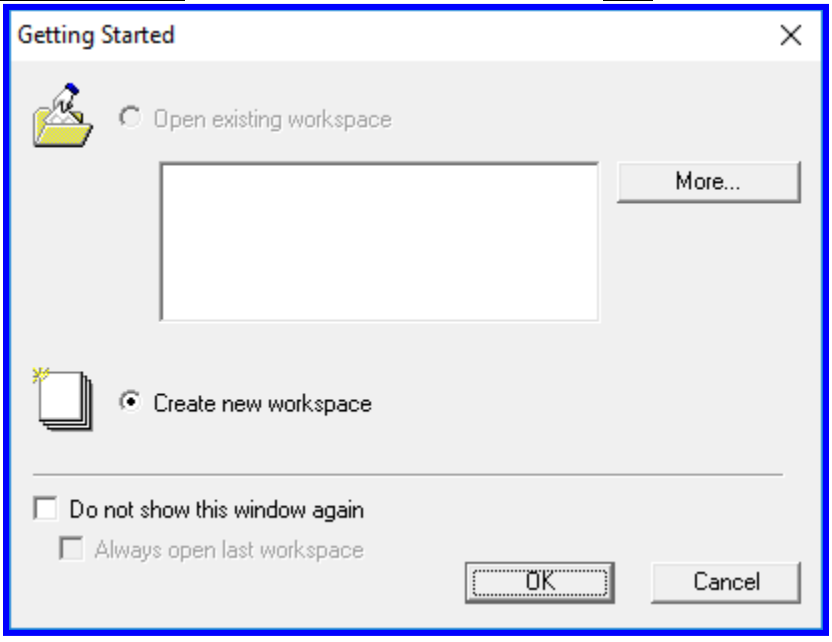
This tutorial will guide you through:

- Setting up a new workspace
- Using the State Diagram wizard to create a state diagram, including the definition of states, conditional transitions, and outputs.
- Simulating the design using stimulators (clock waveforms in this example) and verifying that the output waveforms are correct.
- Using the created sequential circuit as a component. This component, along with a component to create a 1 Hz clock and a component for a BCD to 7-segment decoder, will be incorporated into an overall VHDL file to be synthesized and implemented on the BASYS3 FPGA board.

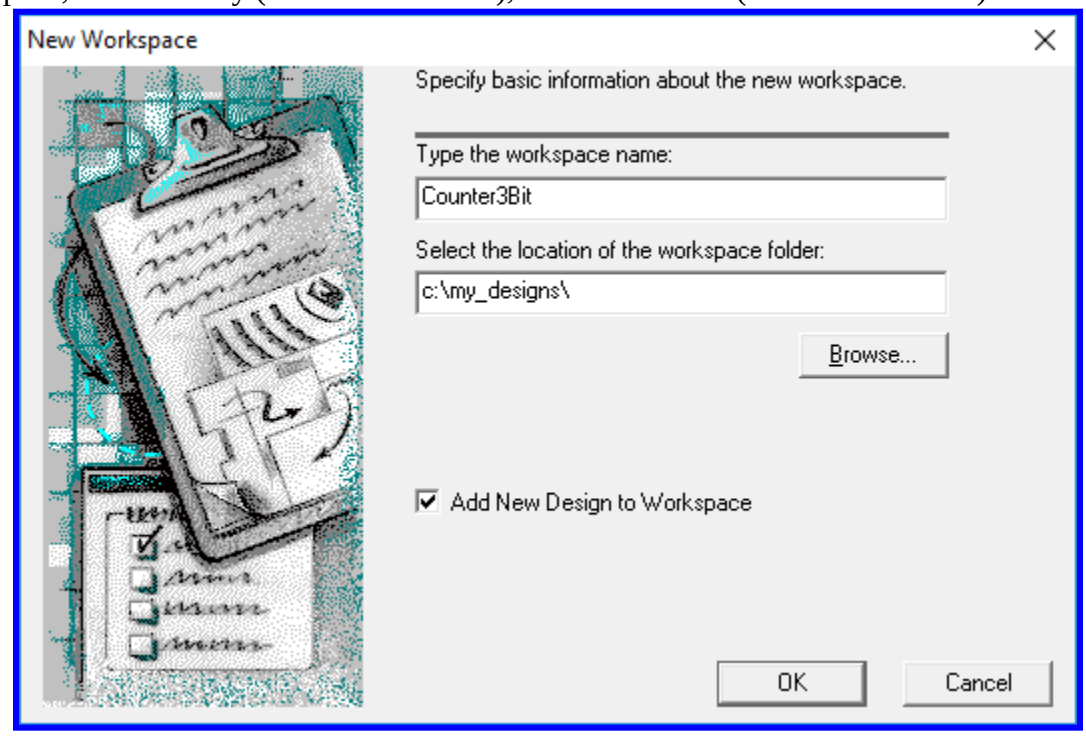
Refer to the tutorial **"Tutorial 1: Combinational Logic Circuits Using Aldec Active-HDL and Xilinx Vivado**" for synthesizing and implementing a design on the BASYS3 FPGA board.

1. Creating a Project with Aldec Active-HDL

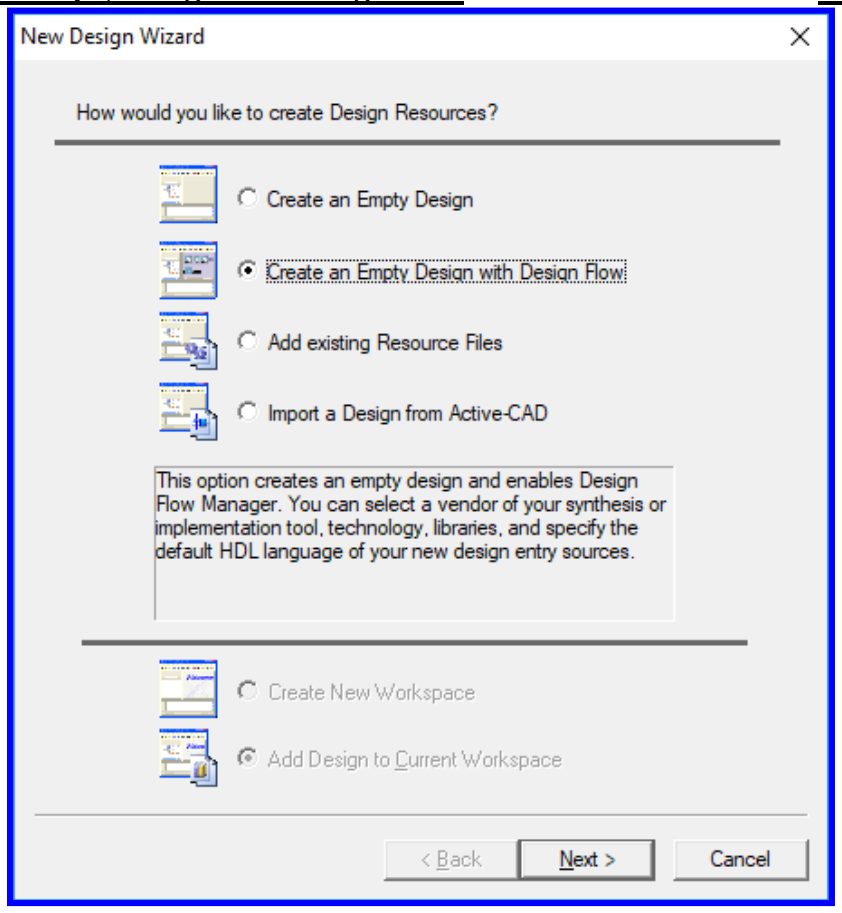
- Launch **Aldec Active-HDL**
- If licensing windows appear, select **Next** until the Getting Started window appears as shown below.
- Select **Create new workspace** as shown below and then select **OK**.



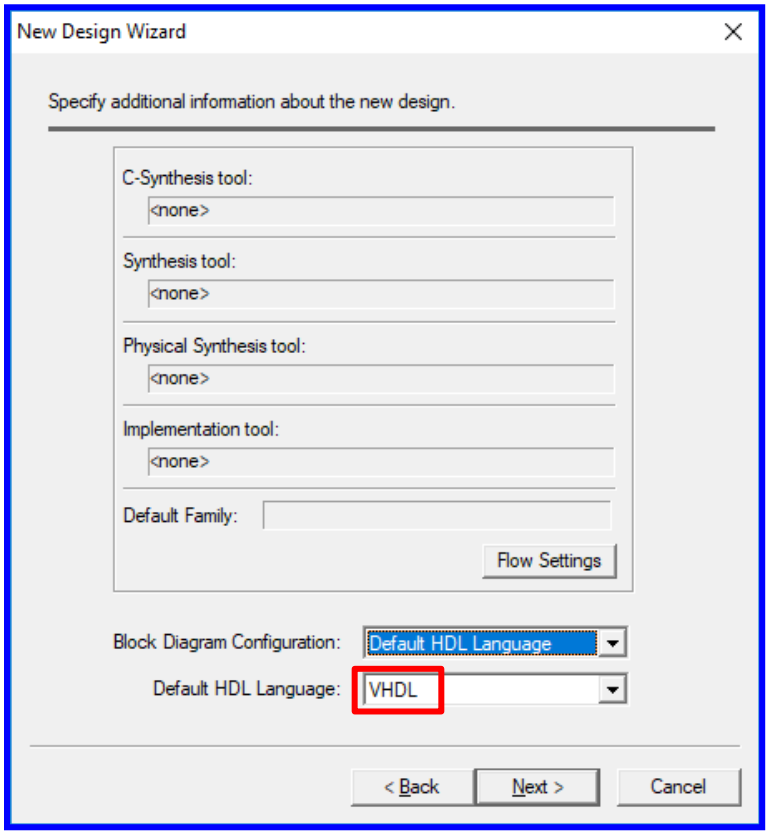
- Enter a name for the workspace (**Counter3Bit** was selected below), change the location of the workspace folder (or use the default as below), and select **OK**. Note that the name for the project workspace, VHDL entity (to be entered later), and architecture (to be entered later) should be the same.



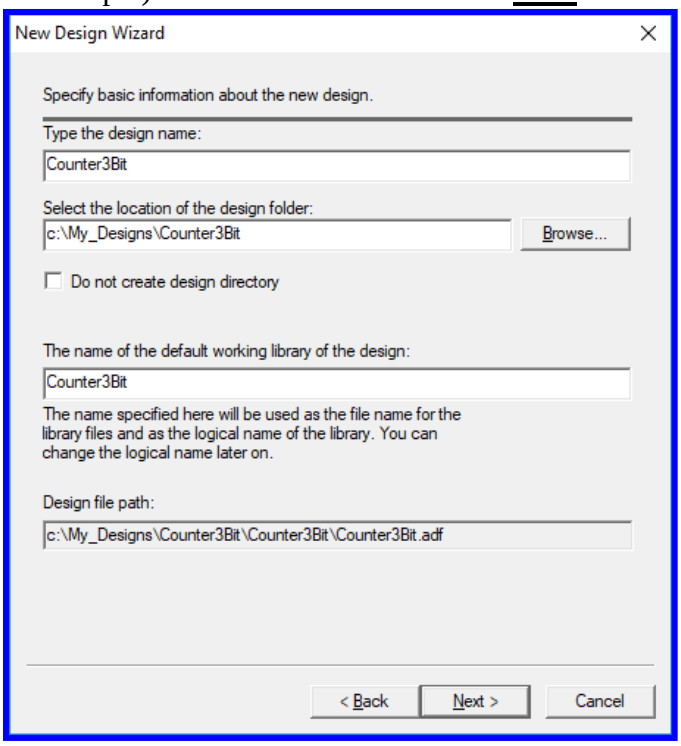
- Select **Create an Empty Design with Design Flow** as shown below and select **Next**.

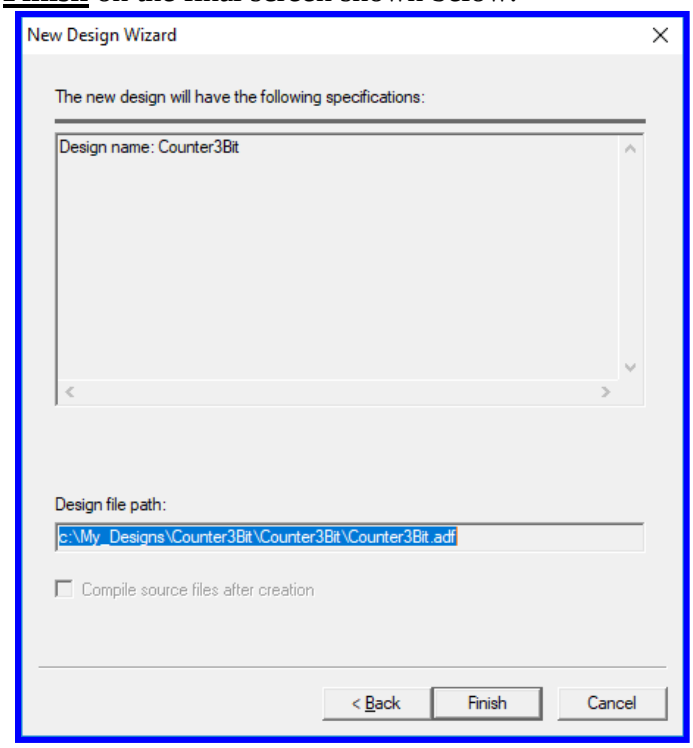


- The next window that appears shows information about synthesis tools and implementation tools that might be configured to launch automatically from Aldec Active-HDL. Synthesis tools can also be launched separately rather than integrating them into Aldec. We will launch Xilinx Vivado separately, so you can ignore most settings shown. Select **VHDL** for the Default HDL Language and then select **Next**.



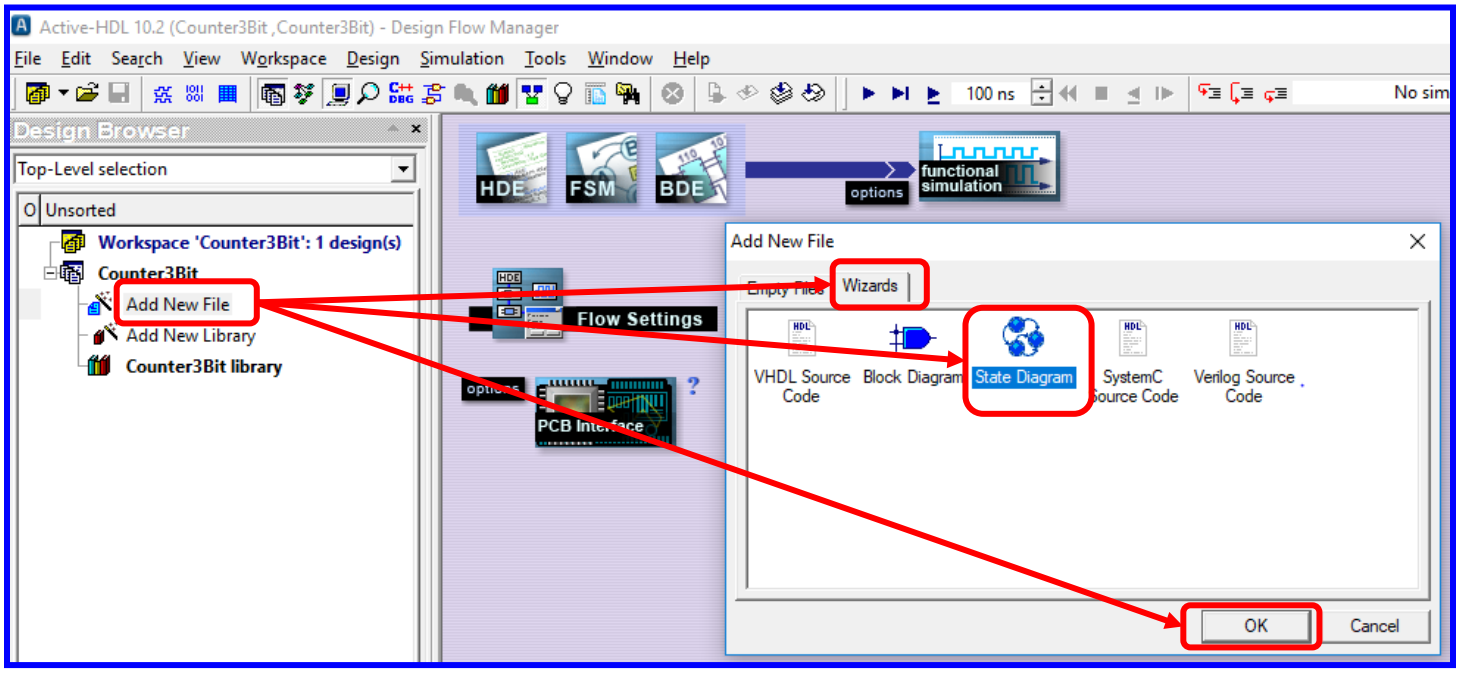
- Enter the design name. Note that it should match the workspace name used earlier (Ex1 for this example). Enter the name and select **Next**. Select **Finish** on the final screen shown below.



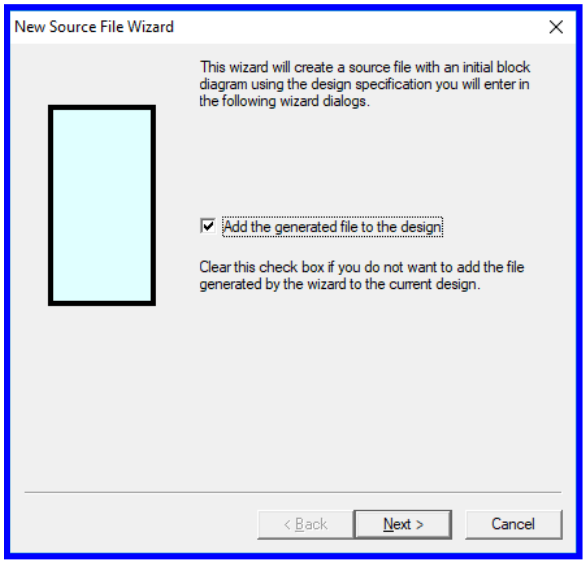


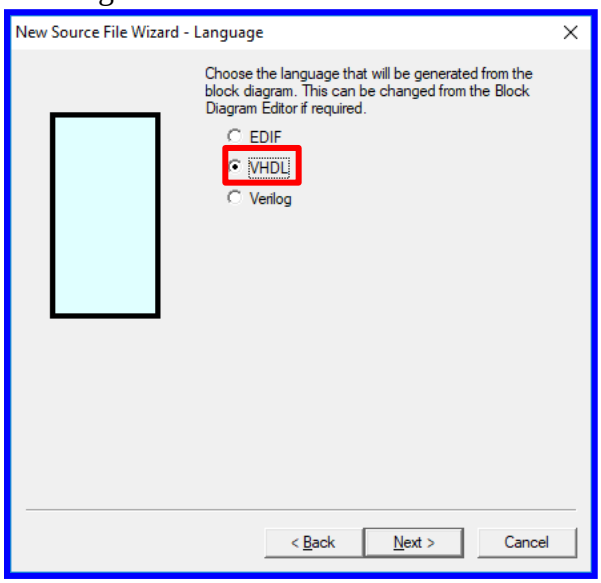
2. Specifying your sequential circuit design using a state diagram

- The Design Flow Manager screen should have appeared after the last step.
- The Design Browser should appear on the left of the screen. If it does not appear, it can be toggled on and off using Alt + 1.
- Double-click on **Add New File** under the Design Browser and the Add New File window should appear.
- Note that there are several types of files that can be specified and that a new file wizard can be used to assist you in specifying inputs and outputs. Select **Wizards**, select **State Diagram**, enter a File Name (**Counter3Bit** in this case) and then select **OK**.

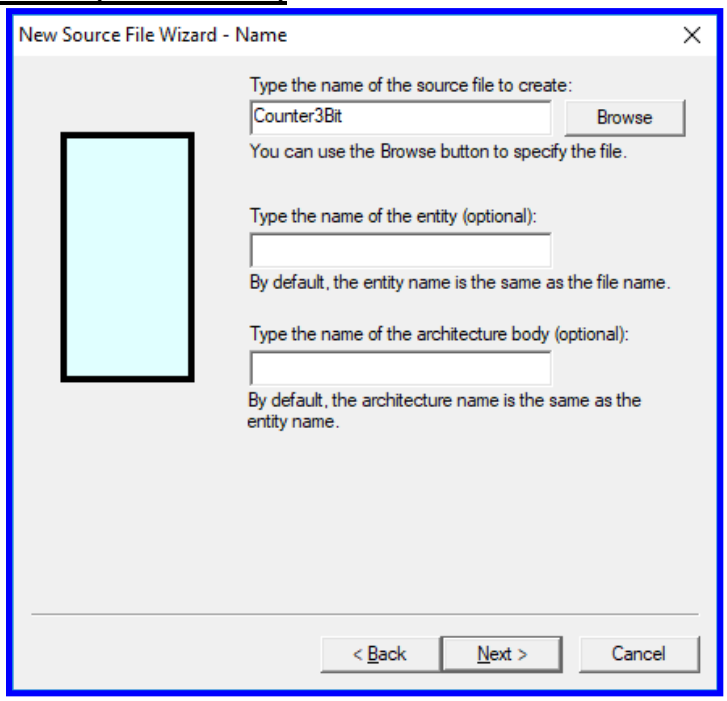


- Select **Next** in the screen below on the left.
  Select **VHDL** and then **Next** in the screen below on the right.

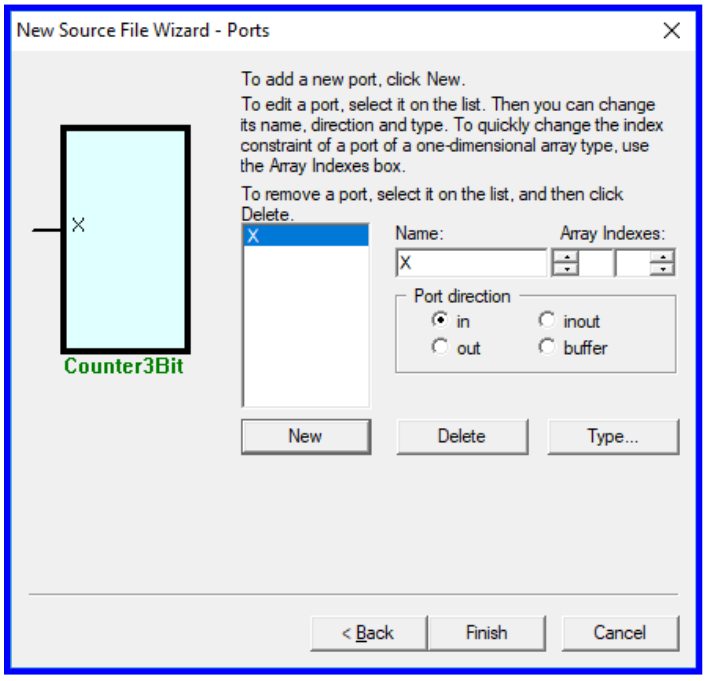


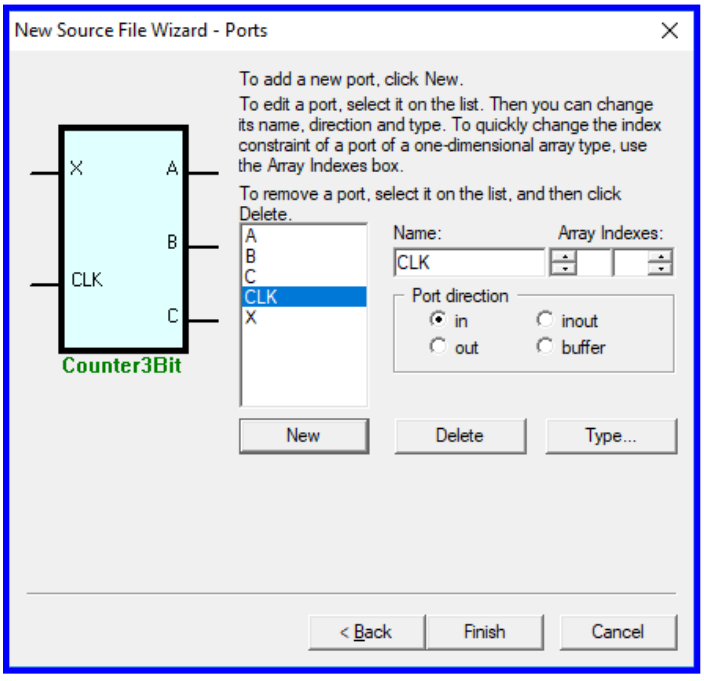


- Enter the **source file name (Counter3Bit)** in the window shown below and select **Next**.

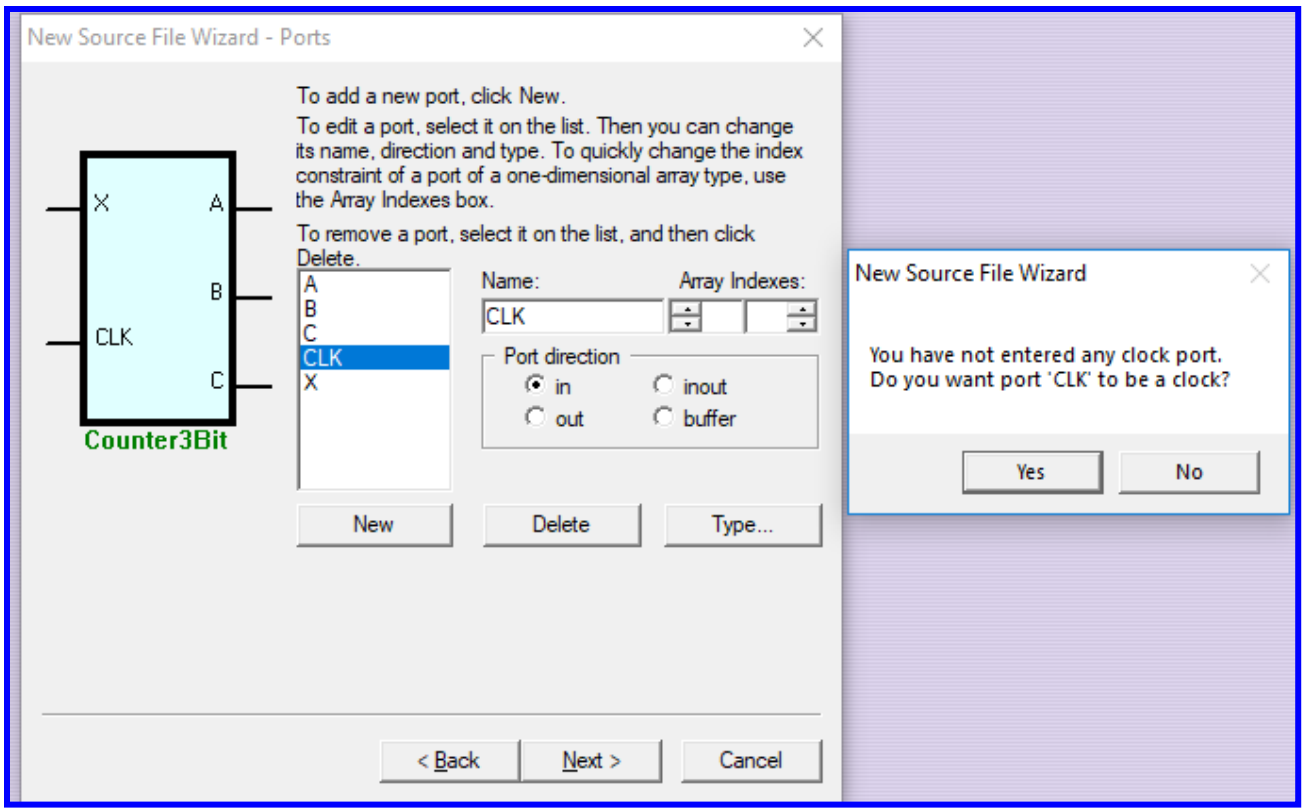


- Before we proceed, it is a good idea to be clear on exactly what inputs and outputs are needed for a 3-bit up/down counter. The counter will require a clock input, a count direction control (x), and a 3-bit output (ABC where A is the MSB).
- Select **New** in the window below on the left to add a new port. Name it **X** with direction **in**.
- Also add input port **CLK** as well as output ports A, B, and C as shown below and then select **Next**.





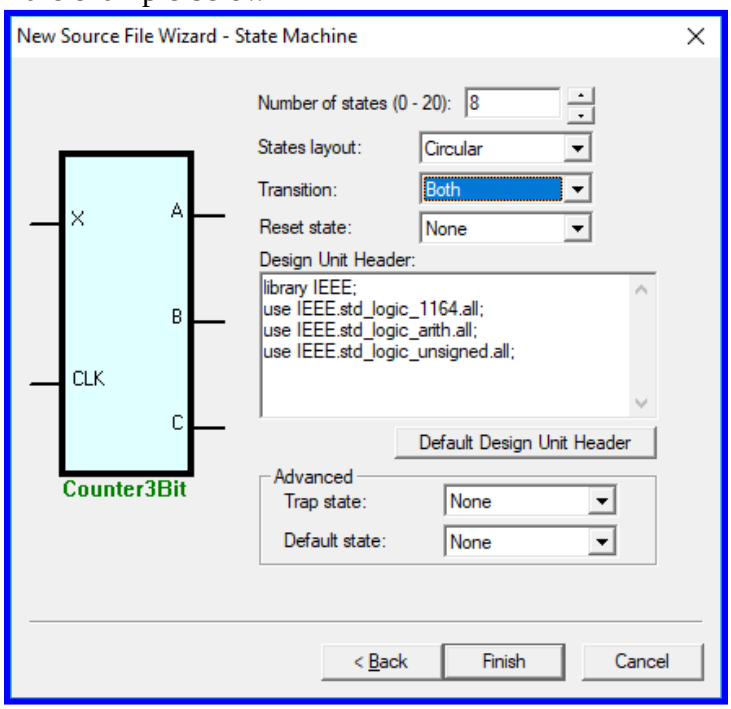
- Note that after selecting **Next** in the screen above, a message appears (shown below): ***"You have not entered any clock port. Do you want port CLK to be a clock?"*** Select **Yes**.



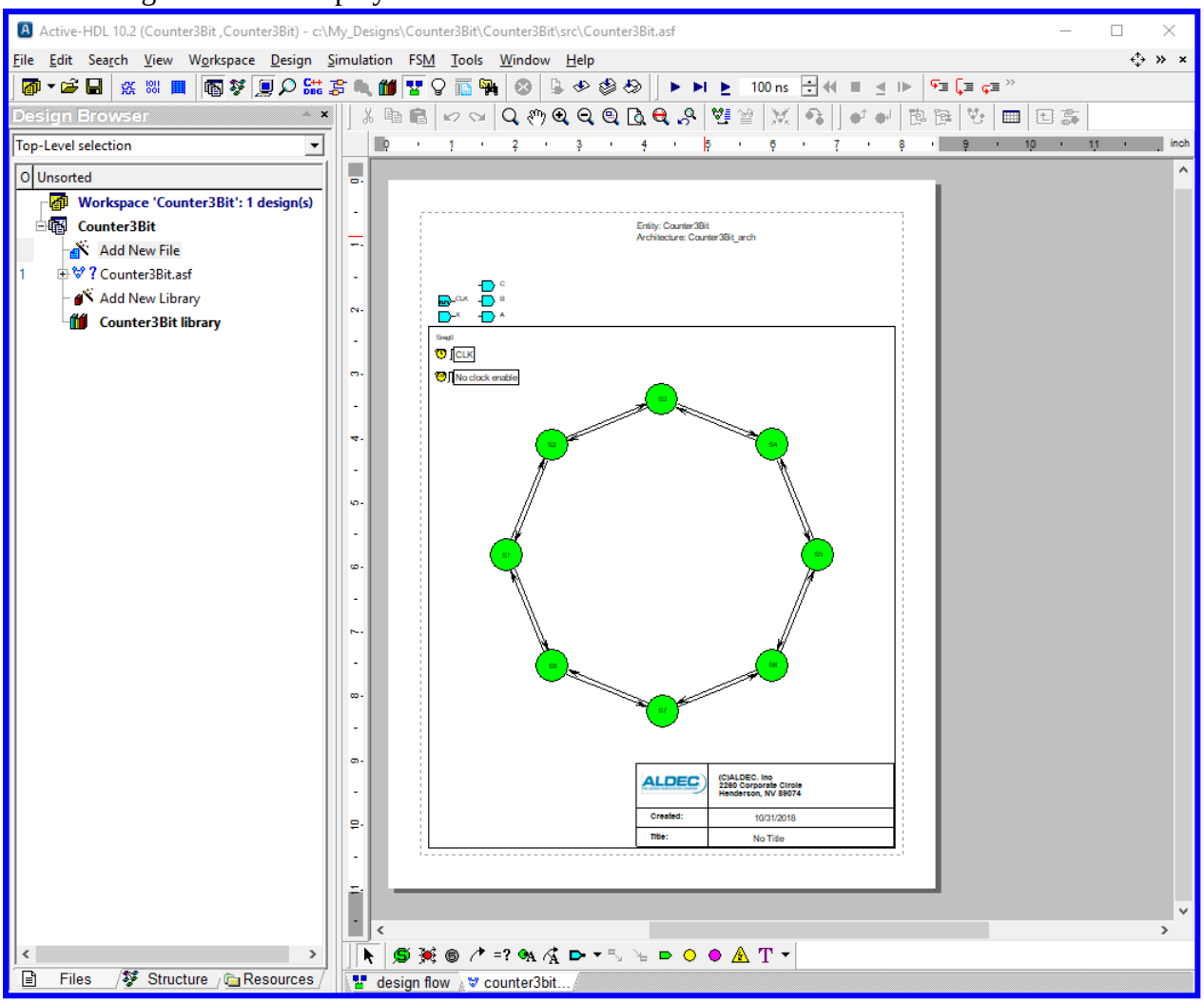
- Before proceeding the state diagram to be implemented must be known. In this example a 3-bit (mod-8) up/down counter is to be implemented. In input switch X will be used where the counter will count UP when X=1 and the counter will count DOWN when X=0. So the desired state diagram is:

Add your state diagram here (by hand)

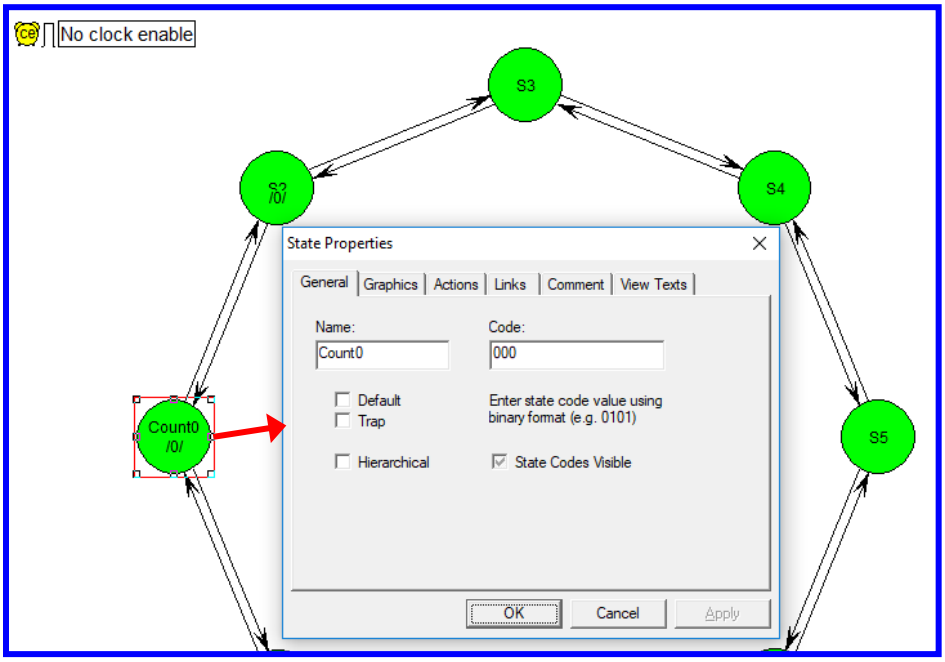
- Individual states and state transitions can be added in Aldec, but the State Machine wizard will do a lot of the work for you. In the example below



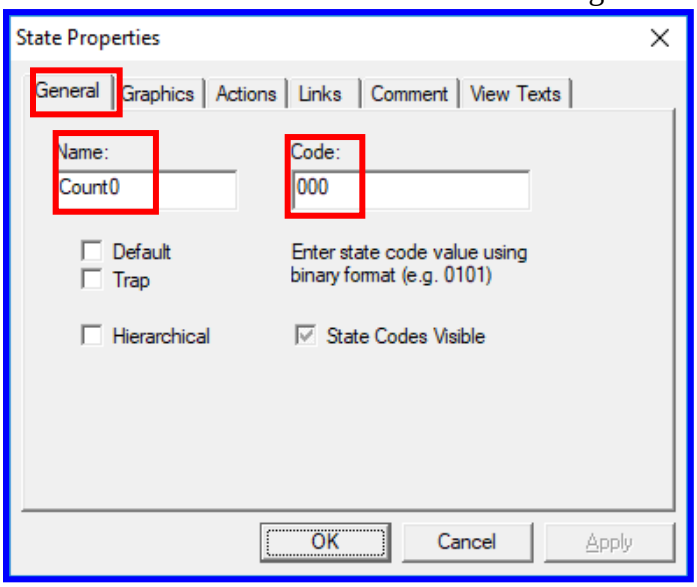
- The state diagram is now displayed as shown below.

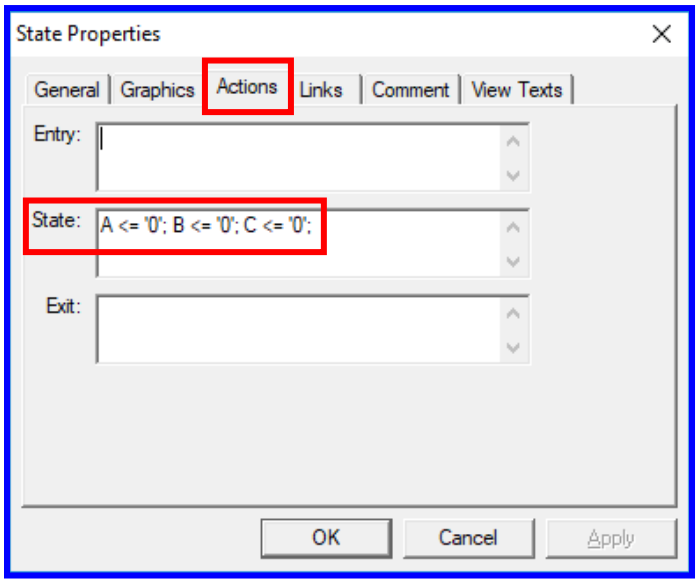


- Zoom in on the state diagram shows it more clearly as seen below.
- Note that 8 states were created: S1 – S8.

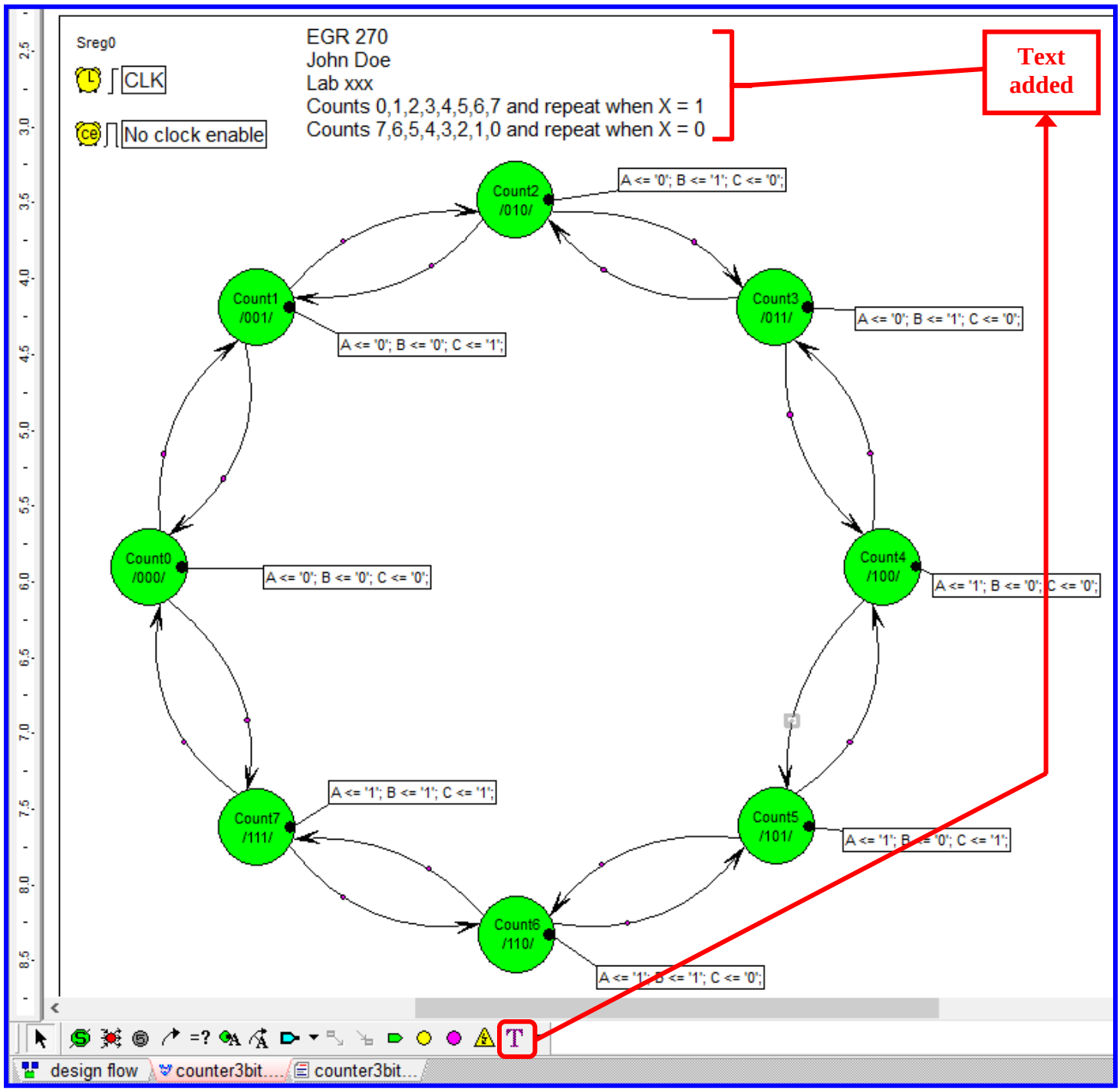


- **<u>Naming States</u>**: Double-click on each state and change the state properties (see above and below).
  - Under the **<u>General tab</u>**:
    - Enter a name for the state (VHDL style) and a binary code.
    - The first state, S1, was renamed as Count0 with the code 000 (see below).
    - The second state, S2, was renamed as Count1 with the code 001.
    - Continue for the remaining states
  - Under the **<u>Actions tab</u>**: Enter the values to be assigned to the outputs (and copy this information so that you can paste it and edit it in the next state).
    - Count0 was given the following actions: A <= '0'; B <= '0'; C <= '0'; (see below)
    - Count1 was given the following actions: A <= '0'; B <= '0'; C <= '1';
    - Continue for the remaining states.

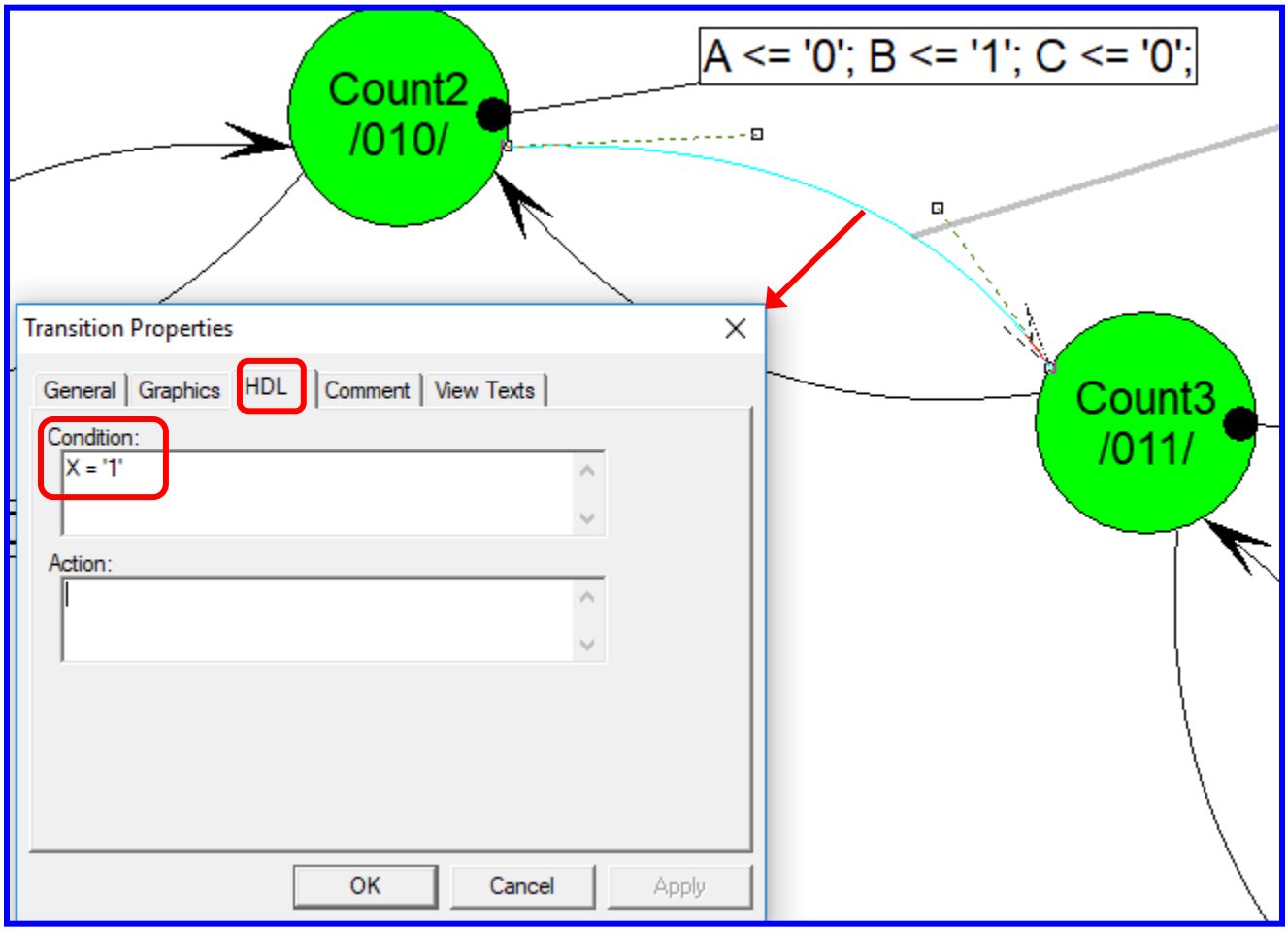




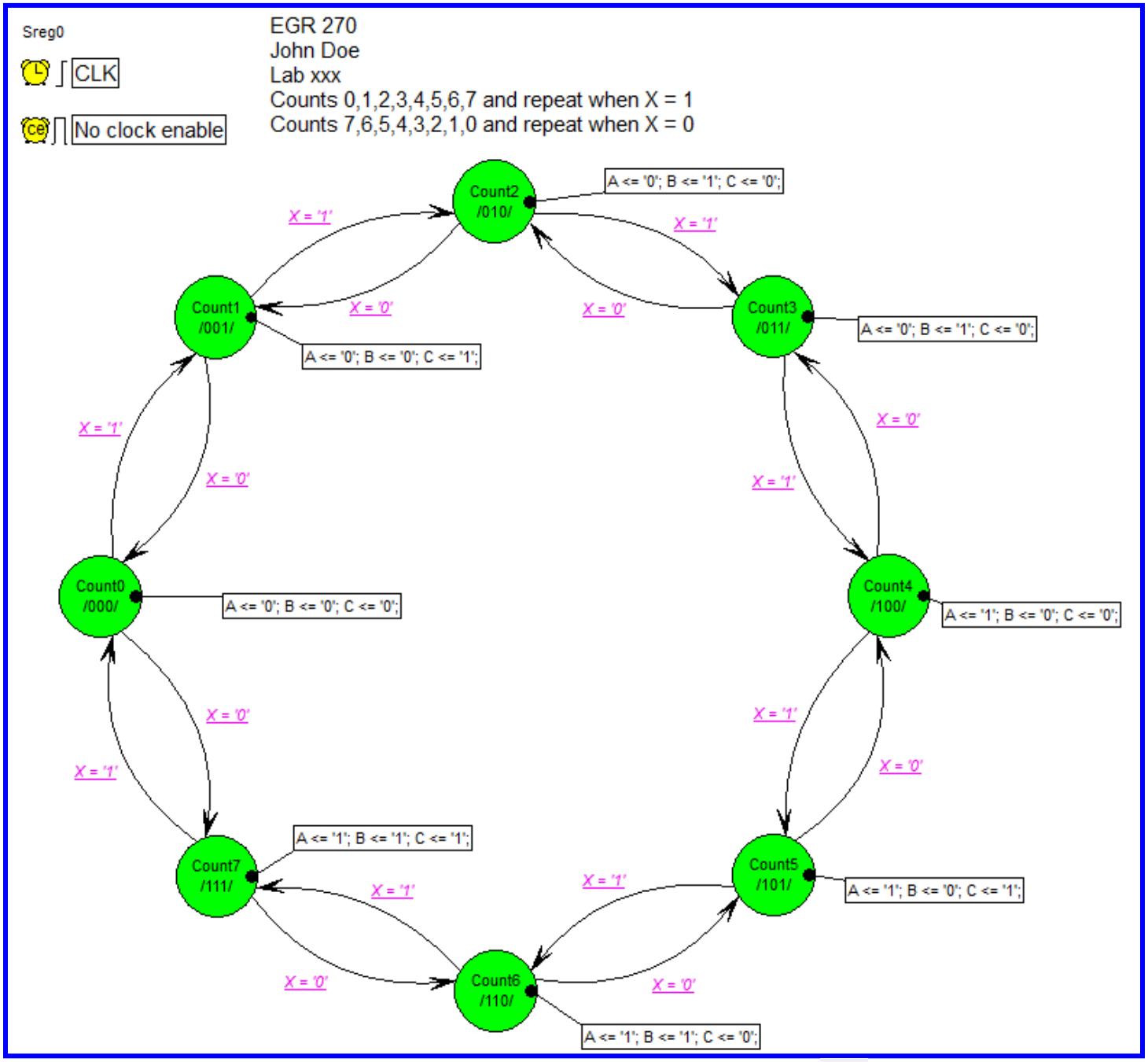
- The state diagram for the 3-bit up/down counter is shown below after naming all states and specifying actions for each state. You might also need to drag states or boxes to new locations to make room.
- **Adding Text**: Note that text was added to the top of the page using the text tool on the toolbar.



- **<u>Adding Conditions to Transitions</u>**: Double-click on the transition line and add the condition under the **<u>HDL tab</u>**
- In the example below, double-clicking on the transition from Count2 to Count3 opened the Transition Properties window. The HDL tab was selected and the condition X=’1’ was added (no semicolon).
- Recall that X=1 to count UP and X=0 to count DOWN.



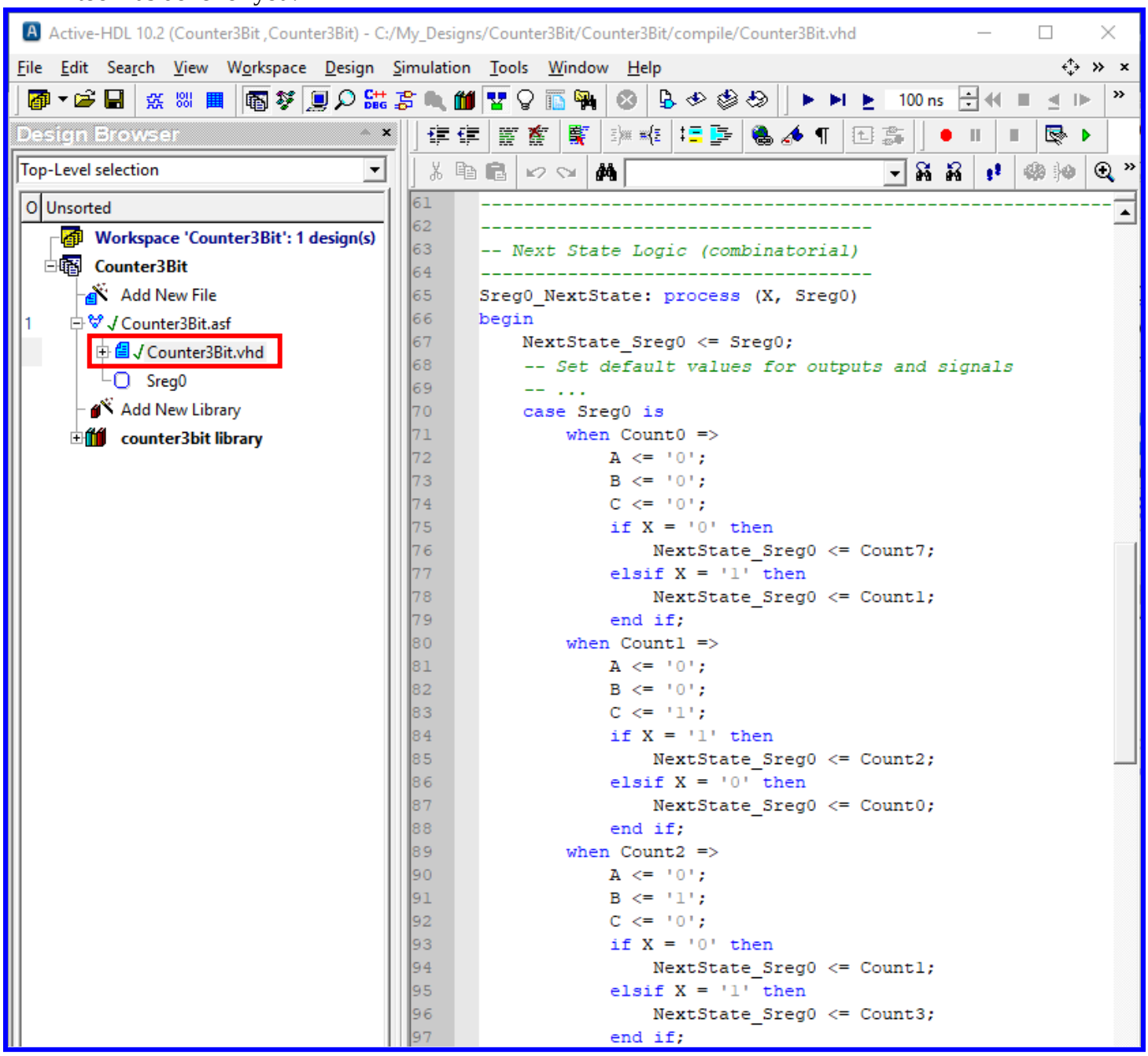
- Continue adding all required transitions.
- Drag the straight transition lines to change them to arcs.
- Carefully drag the transition line conditions and actions as needed so that the state diagram is neat and easy to read.



- Compile your design (Select **Design** – **Compile** or use the Compile tool). Correct any errors that occur.

```
# Compile Entity "Counter3Bit"
# Compile Architecture "Counter3Bit_arch" of Entity "Counter3Bit"
# Compile success 0 Errors 0 Warnings  Analysis time :  0.2 [s]
```

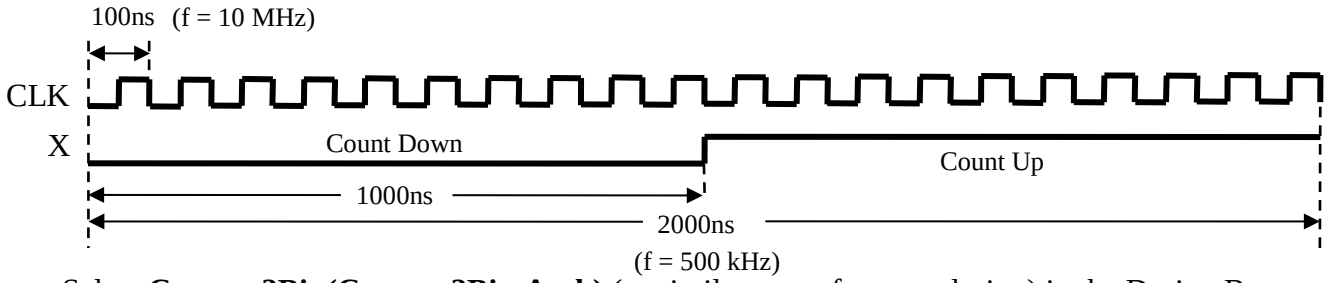
- Note that a VHDL file has been generated with complete entity and architecture sections. Expand the folder Counter3Bit.asf in the browser (or a similar name for your design) to see the VHDL file listed (.vhd). You might want to look at the architecture section to appreciate the work that the Block Diagram tool has done for you!



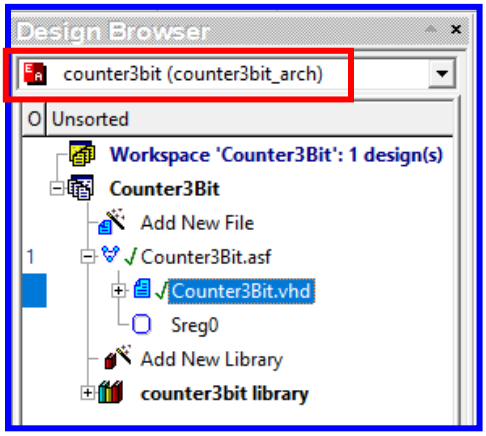
3. Simulating your state machine

- Now we need to simulate the design to see if is correct. We could simulate it as we did with combinational logic circuits: Use the testbench wizard and add VHDL code to specify input waveforms. An alternate way to simulate the circuit is described below.
- First, what input waveforms should we specify? We should clock our 3-bit counter at least 8 times while X = 1 (count up) and at least 8 times while X = 0 (count down). Suppose that we clock it 10 times in each direction for a total of 20 clock pulses. If each clock pulse is 100ns in length then the analysis should last 2000ns and the clock has a frequency of 1/100ns = 10 MHz. Additionally, we could use another clock for X with a period of 2000ns (i.e., low for the first 10 counts or 1000ns and HIGH for the

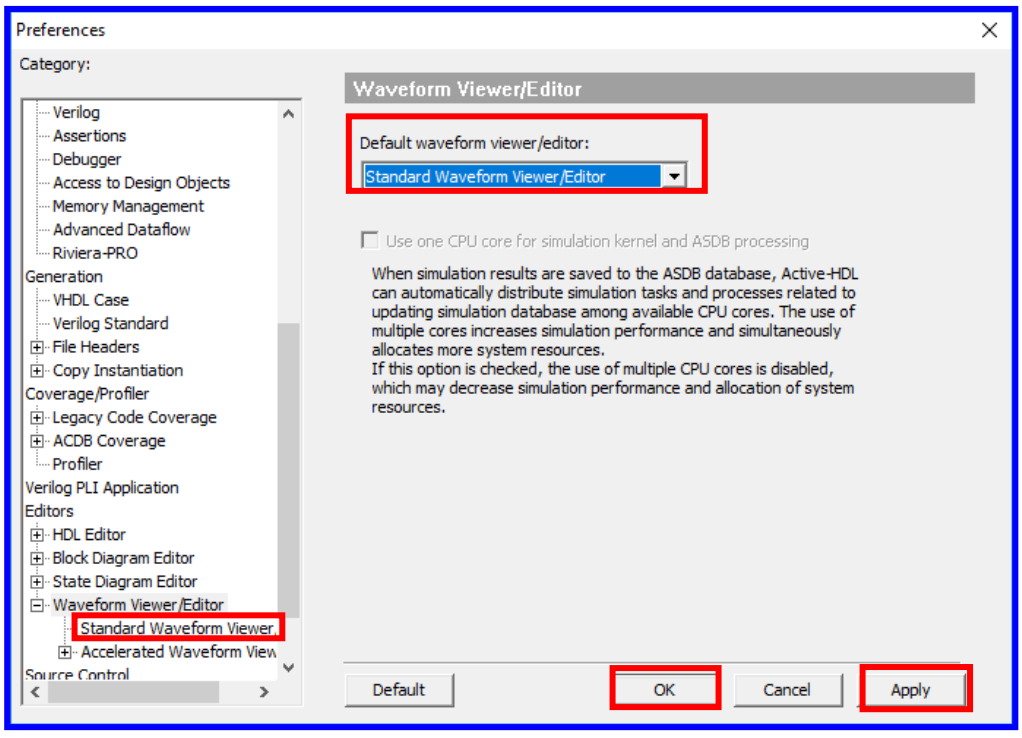
next 10 counts), so the clock frequency for X would be 1/2000ns = 500 kHz. These waveforms are illustrated below.



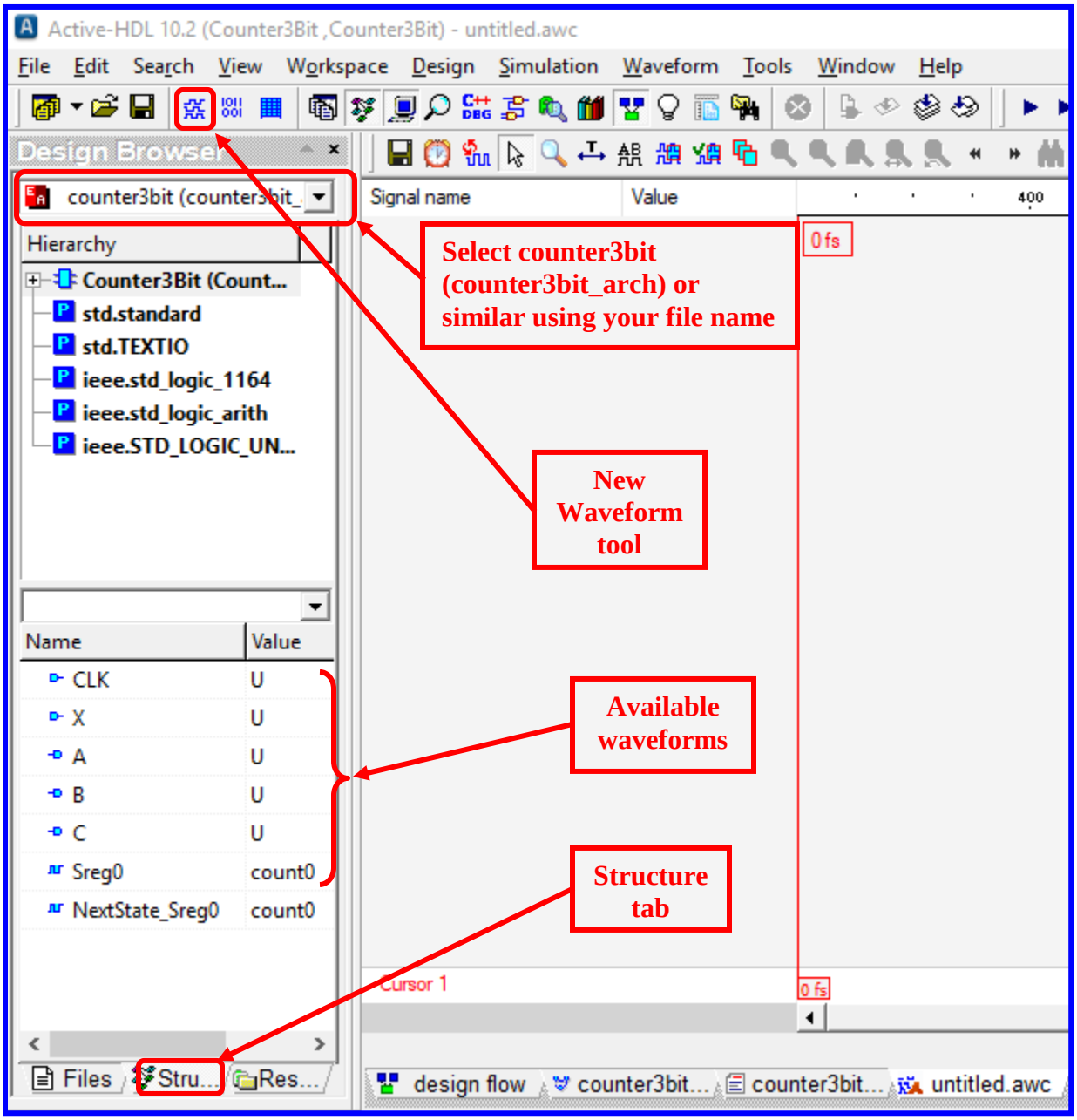
- Select **Counter3Bit (Counter3Bit_Arch)** (or similar name for your design) in the Design Browser as shown below.



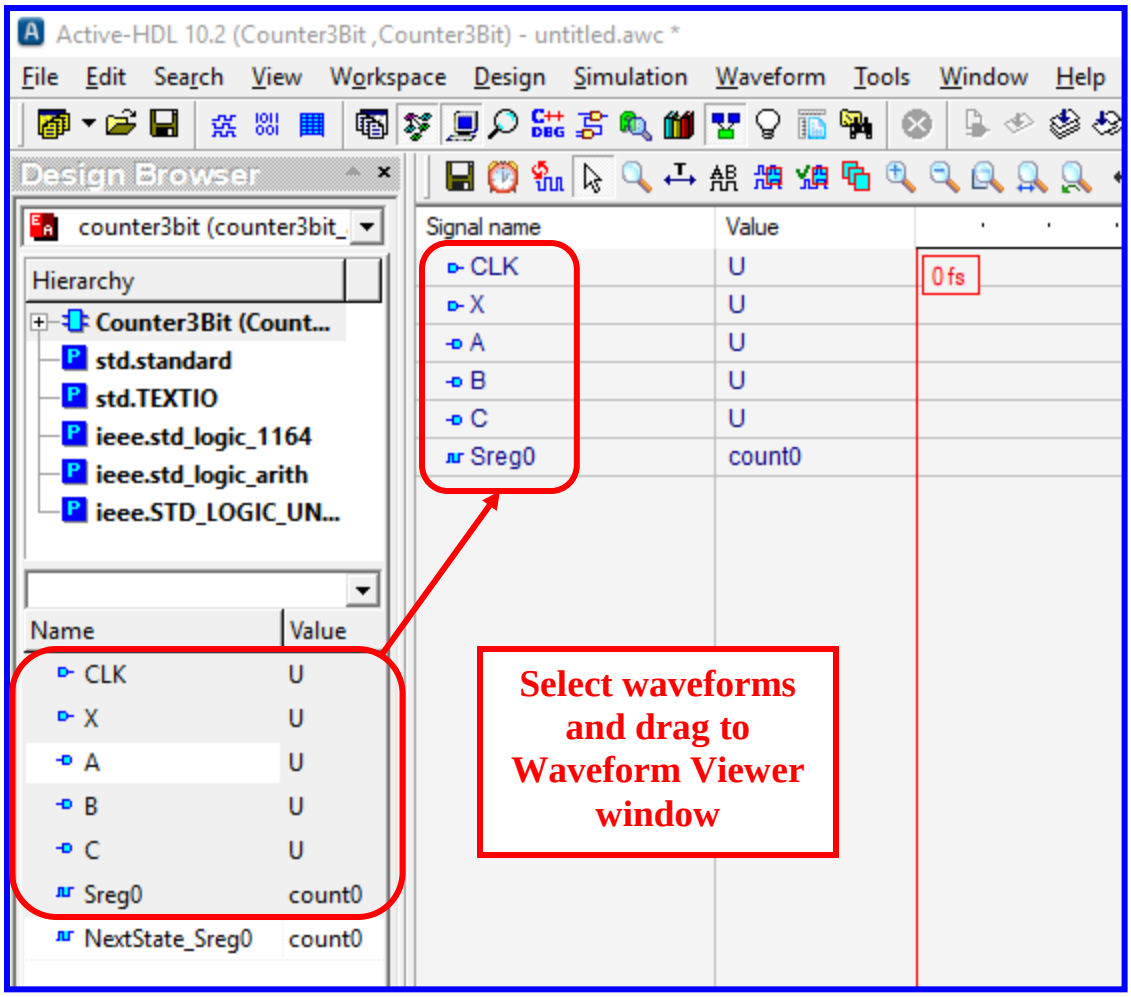
- Before we display waveforms, be sure to set the default waveform viewer as follows:
- Select **Tools – Preferences** and the window below should appear. Select Waveform Viewer/Editor from the Categories listed in the left part of the window, change the default waveform viewer/editor to **Standard Waveform Viewer/Editor** and select **Apply** and then select **OK**.



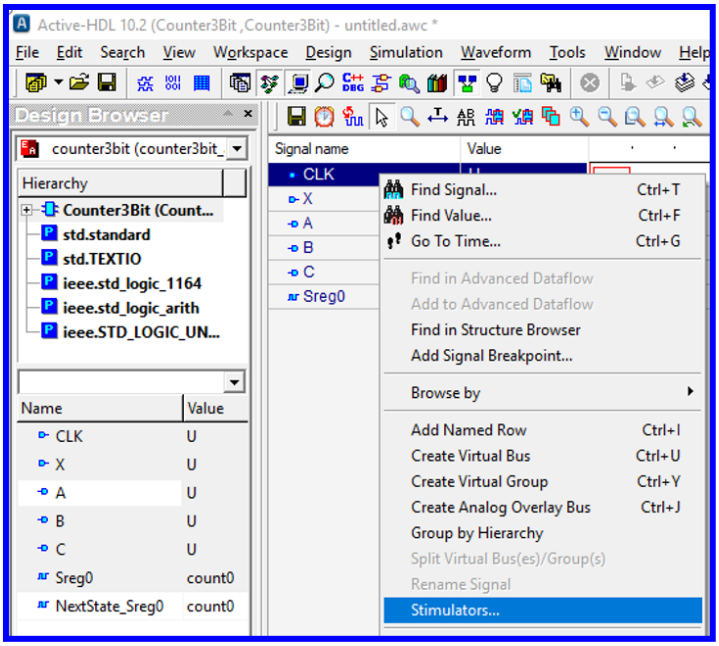
- Select **Initialize Simulation** from the **Simulation** menu.
- Create a new waveform window by selecting the **New Waveform tool** on the main menu (shown below).
- Select the **Structure tab** in the design browser and select **counter3bit (counter3bit_arch)** or something similar using your file name so that the available waveforms are listed as shown below.



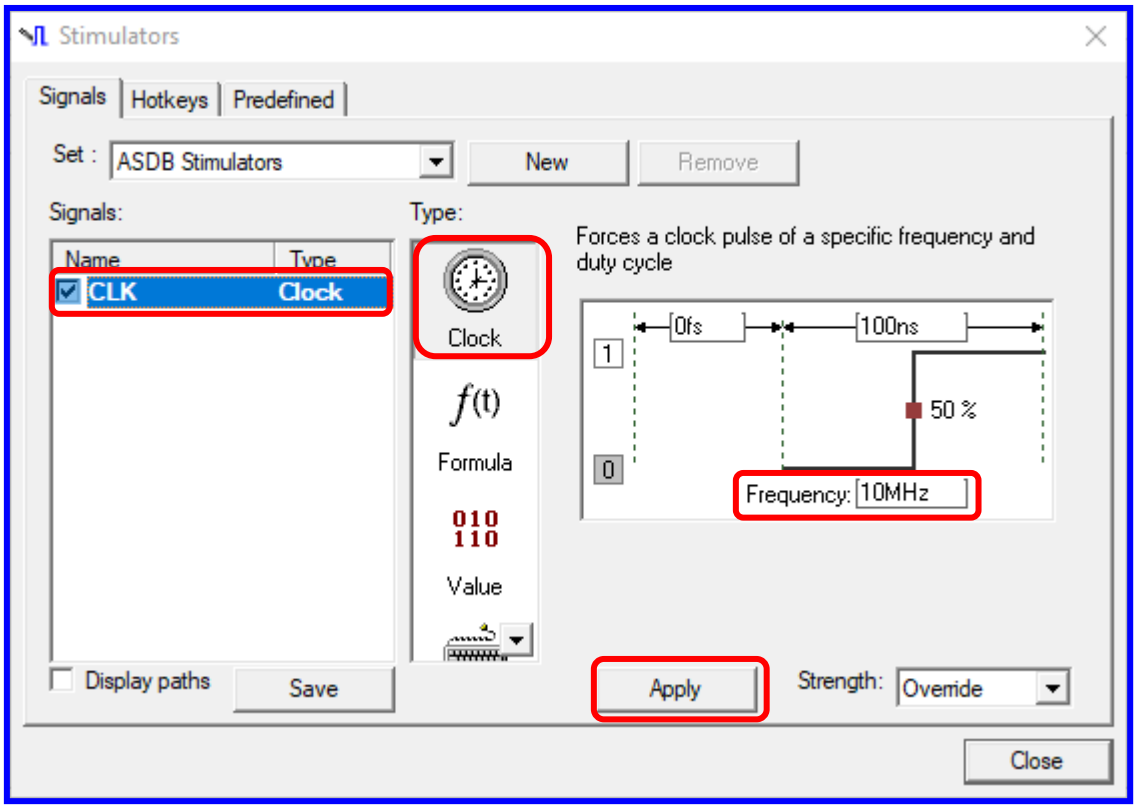
- Select the desired waveforms and drag them to the Waveform Viewer window. (Select any waveform and then used Ctrl + A to select all waveforms.) Drag the waveforms to rearrange them in the desired order if necessary.



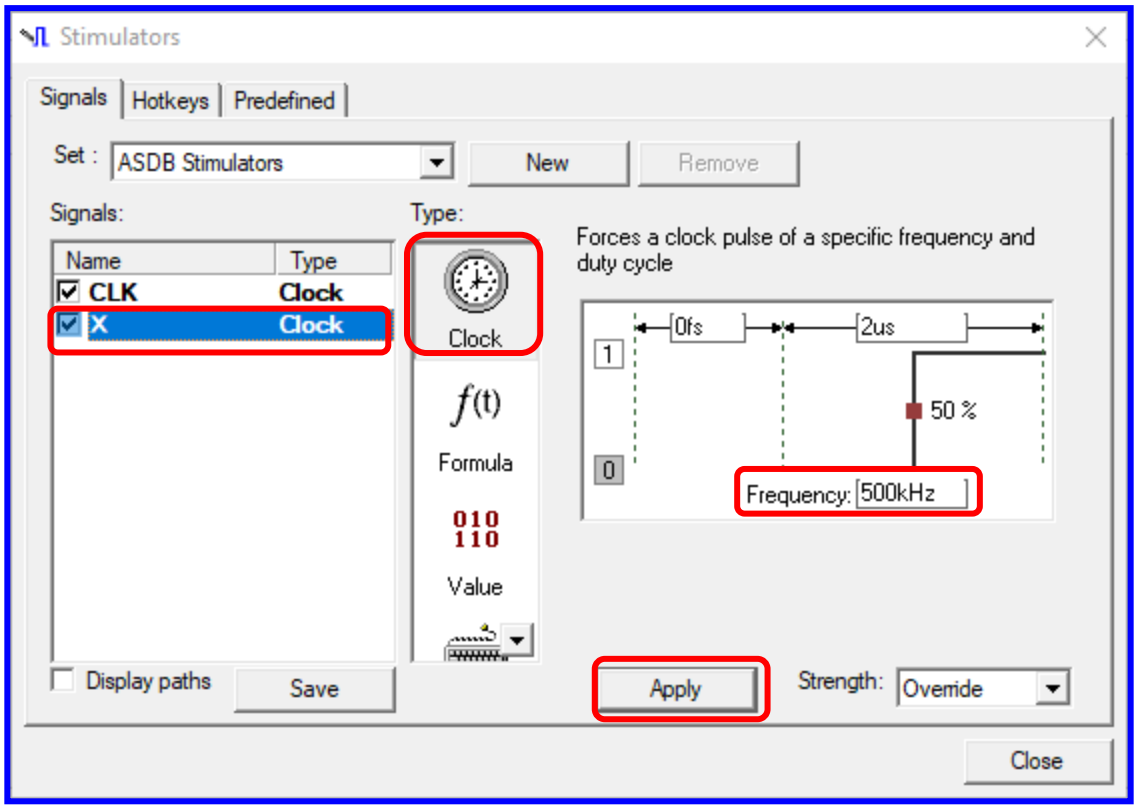
- Right click on waveform CLK and pick **Stimulators…** from the menu that appears.



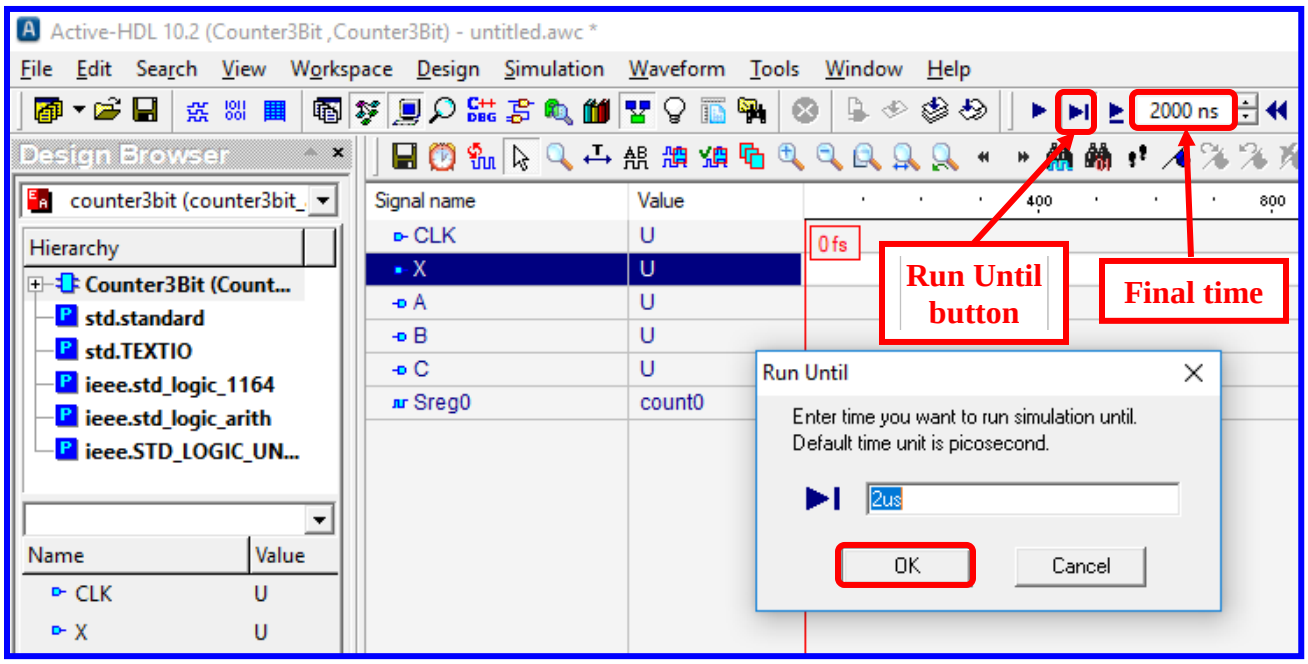
- Click on **Clock**, set the frequency to 10 MHz, and select **Apply**.



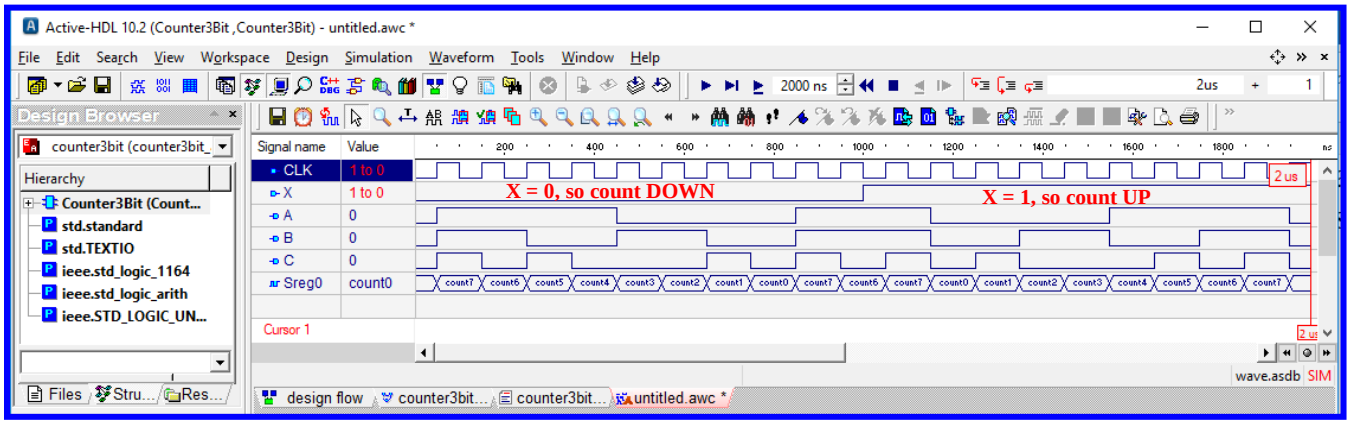
- Similarly, right-click on waveform X, select **Stimulators**, select **Clock**, set the frequency to 500 kHz, and select **Apply**.



- Set the final time on the main menu to **2000ns** and select the **Run Until** button. The Run Until window will appear as shown below. Select **OK**.



- The output waveforms A, B, and C should now be correct. Study the waveforms and note that initially X = 0 and the count is 0, 7, 6, 5, 4, 3, 2, 1, 0, 7, 6 and then when X = 1 the count is 6, 7, 0, 1, 2, 3, 4, 5, 6, 7, 0. Include the signal Sreg0 so that the name of each state will be displayed.
- Use ***Zoom to Fit*** to display all counts concisely.

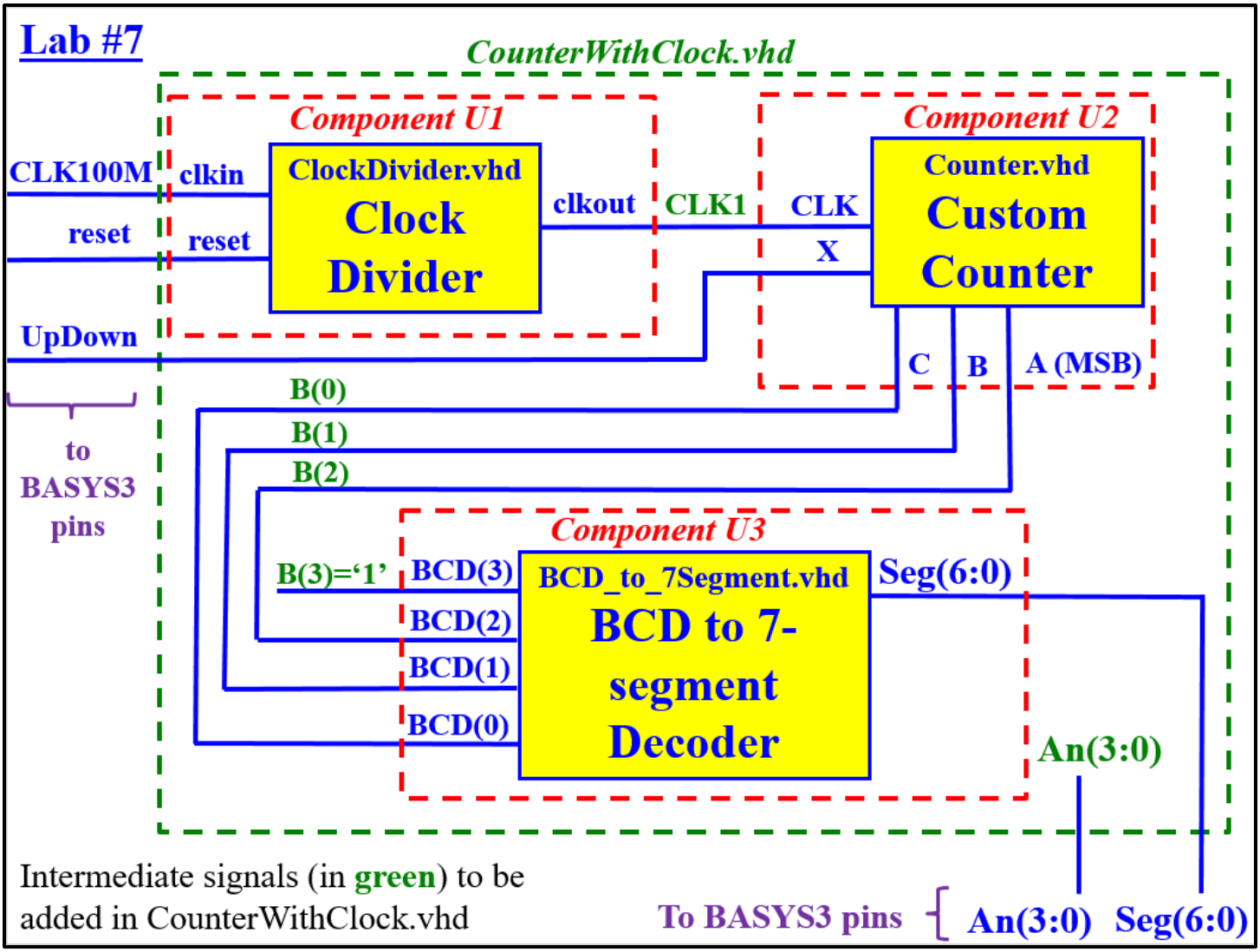


Using Components in VHDL Designs

Now that we have simulated the 3-bit counter and are sure that it is working correctly, we would like to implement design into an FPGA. However, we also need additional items:

- Clock source: a 100MHz internal clock is available on the BASYS3 FPGA board
- Clock divider: we can divide the 100MHz clock by 100E6 to generate a 1 Hz clock
- BCD to 7_segment decoder: the output of our 3-bit counter is in BCD form
- 7-segment display: available on the BASYS3

This is illustrated by the schematic on the following page.



A total of 4 VHDL files will be used:

- **Counter3Bit.vhd** – We just created and tested this file
- **ClockDivider.vhd**
  - Available on the course Bb site
  - Shown on the following pages
  - Divides a 100MHz input clock to produce a 1 Hz output clock
- **BCD-to-7Segment.vhd**
  - Available on the course Bb site
  - Shown on the following pages
  - Converts a BCD value (output of our counter) to common-anode 7-segment display values
- **CounterWithClock.vhd**
  - Available on the course Bb site
  - Shown on the following pages
  - This is the overall VHDL file that uses the other three VHDL files as components. This structural file essentially makes the connections to implement the schematic above.

ClockDivider.vhd

```
-- Clock Divider
-- Divides 100 MHz clock by 100E6 to produce a 1 Hz clock (with a 32.89% duty cycle)
-- Reference:  www.digilentinc.com - Real Digital, Module 10
-- File:  ClockDivider.vhd
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_ARITH.all;
use IEEE.STD_LOGIC_UNSIGNED.all;
----------------------------------------------------------------------------------
entity ClockDivider is
    Port ( clkin, reset : in  STD_LOGIC;
           clkout       : out STD_LOGIC);
end ClockDivider;
----------------------------------------------------------------------------------
architecture ClockDivider of ClockDivider is
constant cntendval : STD_LOGIC_VECTOR(26 downto 0) := "101111101011110000100000000";
-- Note:  100E6 in binary is 101111101011110000100000000
signal cntval : STD_LOGIC_VECTOR (26 downto 0);
begin
    process (clkin, reset)    -- run process whenever clkin or reset change
        begin
            if reset = '1' then cntval <= "000000000000000000000000000";
            elsif (clkin'event and clkin = '1') then
                if (cntval = cntendval) then cntval <="000000000000000000000000000";
                else cntval <= cntval + 1;
                end if;
            end if;
    end process;
clkout <= cntval(26);     -- Concurrent statement, so clkout changes when cntval changes
-- clkout =0 for cntval = 000000000000000000000000000 to 011111111111111111111111111
-- clkout =1 for cntval = 100000000000000000000000000 to 101111101011110000100000000 - 1
-- so clkout = 0 for 67108864 pulses and clkout = 1 for 32891136 pulses
-- for a duty cycle of (32891136/100000000)*100 = 32.89%
end ClockDivider;
----------------------------------------------------------------------------------
```

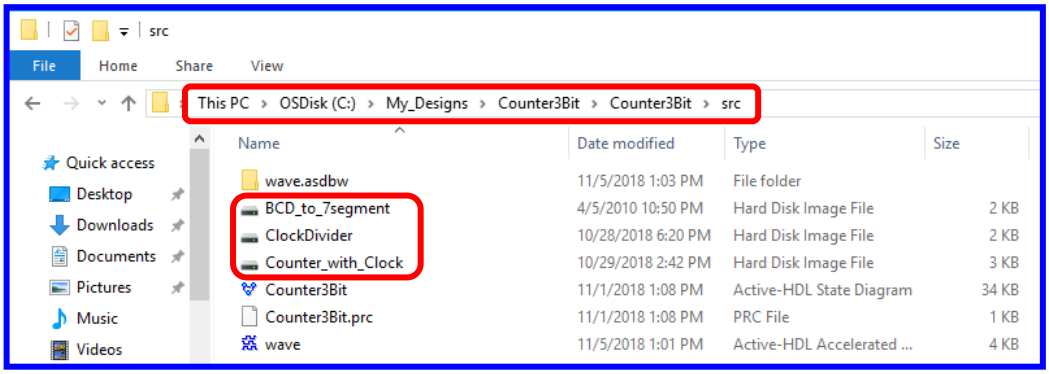
BCD_to_7segment.vhd

```
-- BCD to 7-segment decoder
-- File: BCD_to_7segment.vhd
library IEEE;
use IEEE.STD_LOGIC_1164.all;
---------------------------------------------------------------------------------------------------
entity BCD_to_7segment is
     port(BCD : in STD_LOGIC_VECTOR(3 downto 0);  -- BCD(3) = MSB
          Seg : out STD_LOGIC_VECTOR(6 downto 0)); -- Seg(6) = cathode a
end BCD_to_7segment;                               -- Seg(0) = cathode g
---------------------------------------------------------------------------------------------------
architecture BCD_to_7segment of BCD_to_7segment is
begin
    DisplayEncodeProc : process (BCD)  -- run process whenever input BCD changes
    begin
        case BCD is
            --                      Cathode value (active-Low)
            --                      abcdefg
            when "0000" => Seg <= "0000001";  -- display digit 0
            when "0001" => Seg <= "1001111";  -- display digit 1
            when "0010" => Seg <= "0010010";  -- display digit 2
            when "0011" => Seg <= "0000110";  -- display digit 3
            when "0100" => Seg <= "1001100";  -- display digit 4
            when "0101" => Seg <= "0100100";  -- display digit 5
            when "0110" => Seg <= "0100000";  -- display digit 6
            when "0111" => Seg <= "0001111";  -- display digit 7
            when "1000" => Seg <= "0000000";  -- display digit 8
            when "1001" => Seg <= "0000100";  -- display digit 9
            when "1010" => Seg <= "1110010";  -- display unique pattern used by 7447 for input 10
            when "1011" => Seg <= "1100110";  -- display unique pattern used by 7447 for input 11
            when "1100" => Seg <= "1011100";  -- display unique pattern used by 7447 for input 12
            when "1101" => Seg <= "0110100";  -- display unique pattern used by 7447 for input 13
            when "1110" => Seg <= "1110000";  -- display unique pattern used by 7447 for input 14
            when others => Seg <= "1111111";  -- all digits turned off for input 15 (or X's, U's)
        end case;
    end process;
end BCD_to_7segment;
```

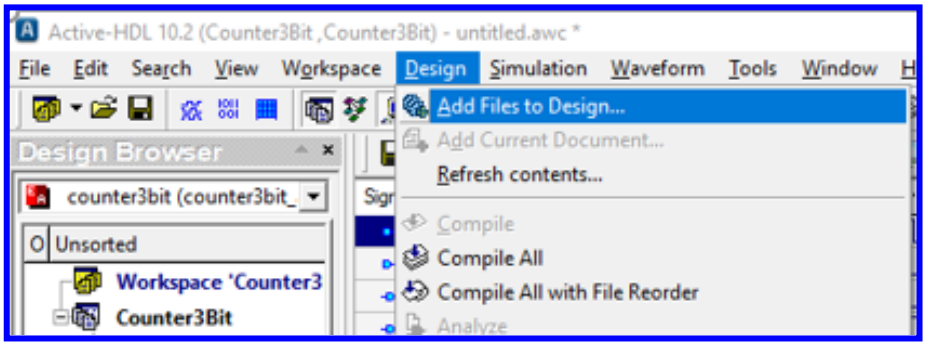
CounterWithClock.vhd

```
-- File:  Counter_with_Clock.vhd
-- Structural file for Lab 7 that uses 3 components
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
-----------------------------------------------------------------------------------------------
entity counter_with_Clock is
    port (
      CLK100M, UPDOWN, reset    : in STD_LOGIC;    -- 100 MHz clock input from BASYS3 (pin W5)
      An    : out STD_LOGIC_VECTOR(3 downto 0);    -- Enable lines to four 7-segment displays on BASYS3
      Seg   : out STD_LOGIC_VECTOR(6 downto 0));   -- Seg(6) = cathode a, ..., Seg(0) = cathode g
end counter_with_Clock;
-----------------------------------------------------------------------------------------------
architecture counter_with_Clock_arch of counter_with_Clock is
-- component declarations
component ClockDivider  -- use component defined in file Clock1Hz.vhd to create 1Hz clock
    port (
        clkin       : in STD_LOGIC;    -- 100 MHz clock available on BASYS2 FPGA board
        reset       : in STD_LOGIC;    -- used to reset the clock divide counter to 0
        clkout      : out STD_LOGIC); -- 1 Hz clock output used to clock 7-seg displays
end component;
component counter3bit  -- use component defined in file counter3bit.vhd (or student file name)
    port (
        CLK, X      : in STD_LOGIC;     -- 1 Hz clock input -- X = 1 to count UP, X = 0 to count DOWN
        A, B, C     : out STD_LOGIC);   -- counter output (A = MSB)
end component;
component BCD_to_7segment  -- use component defined in file BCD_to_7segment
     port(
        BCD : in STD_LOGIC_VECTOR(3 downto 0);  -- BCD(3) = MSB
        Seg : out STD_LOGIC_VECTOR(6 downto 0)); -- Seg(6) = cathode a, ..., Seg(0) = cathode g
end component;
-----------------------------------------------------------------------------------------------
-- Define intermediate signals
signal CLK1   : std_logic;     -- define intermediate signal between components
signal B      : std_logic_vector(3 downto 0);     -- intermediate signals between counter and BCD_to_7segment
-----------------------------------------------------------------------------------------------
begin
    B(3) <= '0';     -- Only 3 bits used for 3-bit counter, so set 4th bit (MSB) to 0
    AN <= "1110";    -- Turn OFF first three 7-segment displays and turn on the 4th display
    -- instantiate components
    U1: ClockDivider       -- define an instance of ClockDivider
    port map(clkin => CLK100M, reset => reset, clkout => CLK1);
    U2: counter3bit        -- define an instance of counter3bit
    port map(CLK => CLK1, X => UPDOWN, A => B(2), B => B(1), C => B(0));
    U3: BCD_to_7segment    -- define an instance of BCD_to_7segment
    port map(BCD => B, Seg => Seg);
end counter_with_Clock_arch;
-----------------------------------------------------------------------------------------------
```

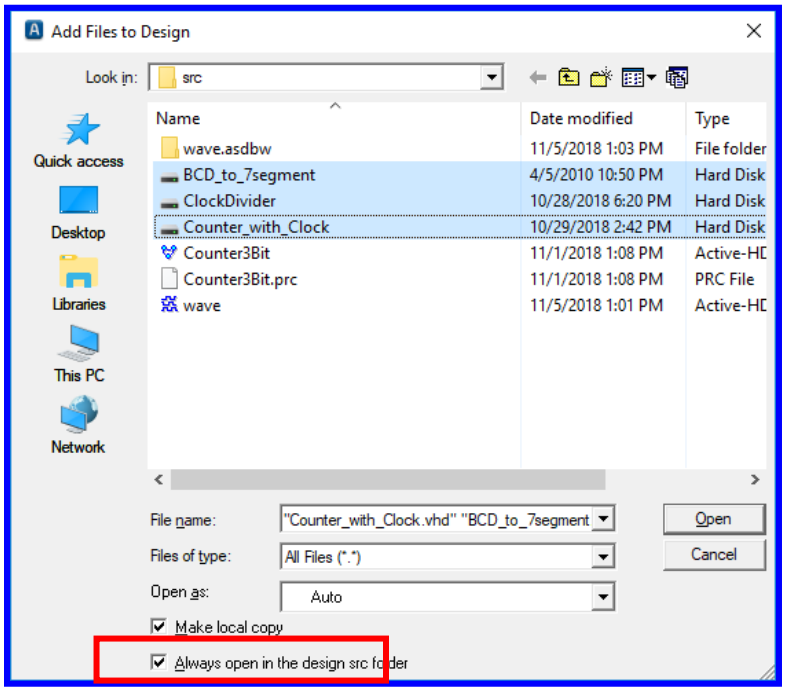
- Download the following files from the course Bb site. They can be stored anywhere, but a good place is to store them in the source (src) folder of the project. (Note that the file Type may not have the correct description and the extension may not be shown, but they will appear correctly within Aldec.)
    - **ClockDivider.vhd**
    - **BCD-to-7Segment.vhd**
    - **CounterWithClock.vhd**



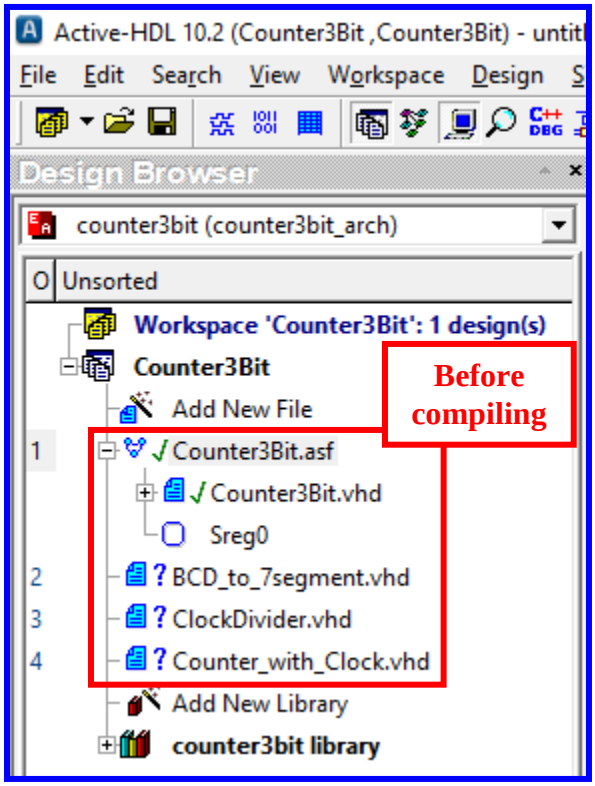
- Add the files to the Aldec Project.
    - Select **Design – Add Files to Design**

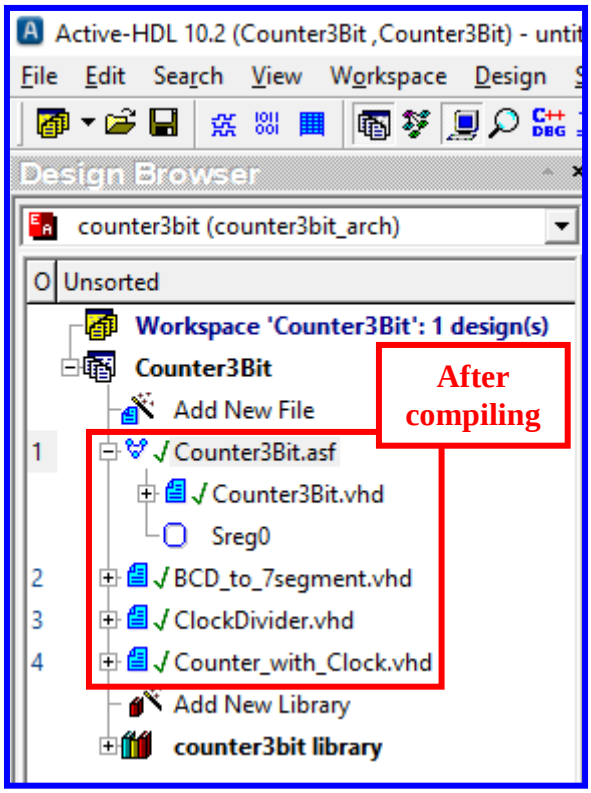


- Locate the files in the project src directory (or wherever you stored them) and select **Open.**



- Compile all of the VHDL files. After adding the files to the project, you should see the files listed in the Design Browser as shown below (Pick the Files tab). Question marks (?) indicate that the files have not yet been compiled.
    - Select **Compile All** to compile the files. Each VHDL file should have a check mark next to it if it compiled successfully. If any files did not compile successfully, you must correct the errors before proceeding.





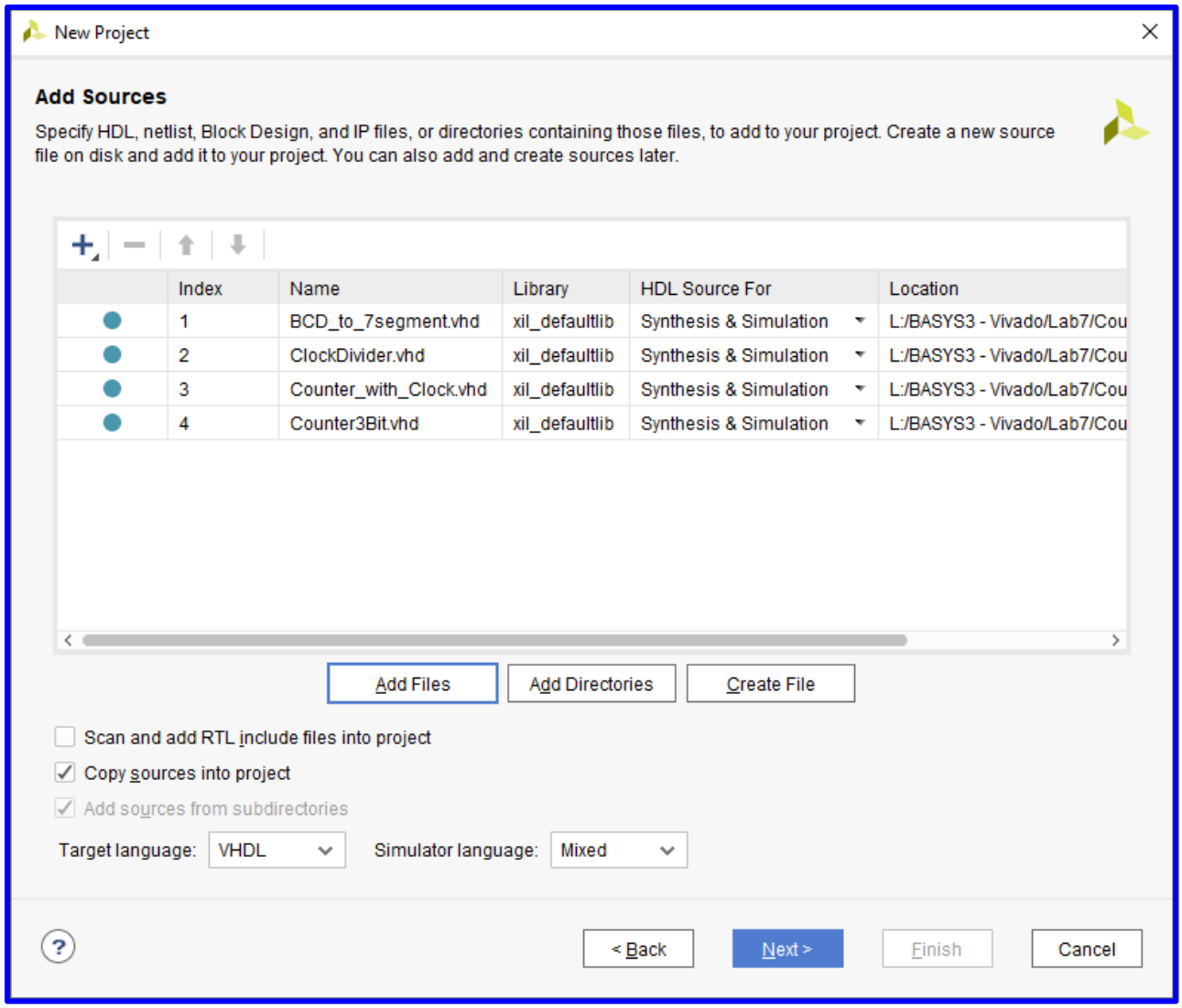
Using Xilinx Vivado to synthesize the design into the Artix-7 FPGA

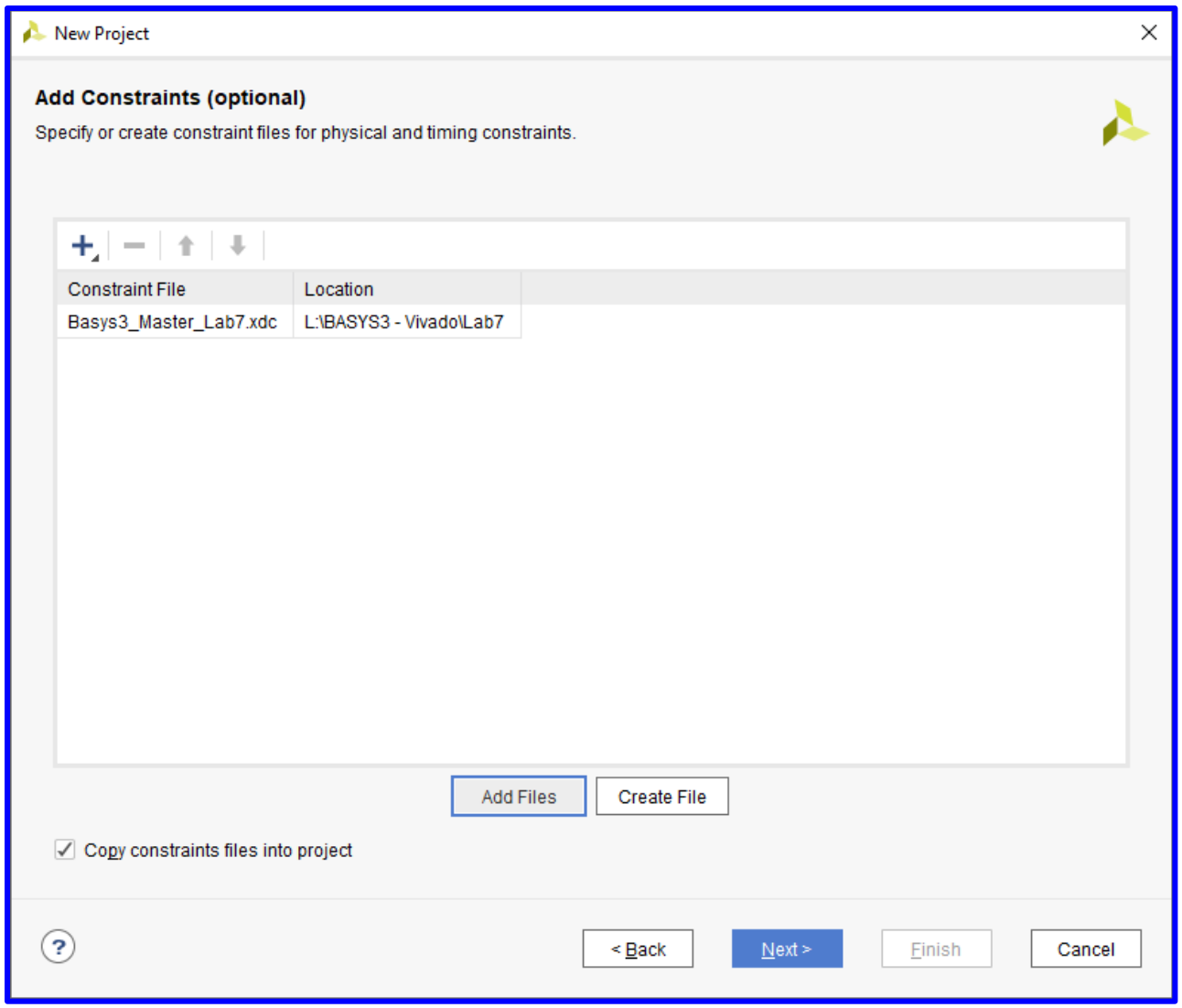
The steps involved here are nearly identical to those used in Lab 5, so refer to the tutorial:

- ***Tutorial 1 - Combinational Logic Circuits using Aldec Active-HDL and Xilinx Vivado***

Some important differences:

- When we **add sources** to our design using the Xilinx software, we need to add all four VHDL files, as shown below.
- Ignore any warnings in Xilinx about excessive delay due to our ClockDivider circuit.
- The final bit file (or bin file) produced should have the name of the overall VHDL file, so when you generate the bitstream, look for the name (in this example): **CounterWithClock.bit** or **CounterWithClock.bin**




New Project
Add Constraints (optional)
Specify or create constraint files for physical and timing constraints.
Constraint File
Location
Basys3_Master_Lab7.xdc
L:\BASYS3 - Vivado\Lab7
Add Files
Create File
Copy constraints files into project
< Back
Next >
Finish
Cancel