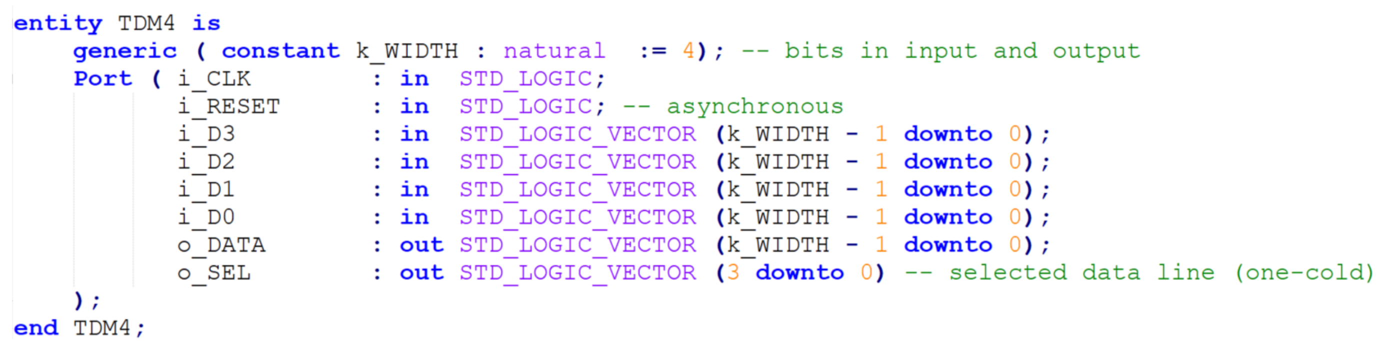
You may seek help from any cadet or instructor, not limited to this course, and reference any publication in its completion. Online resources are acceptable. However, no resources containing a solution is allowed. Your work must **always** be your own. Normal documentation of all resources utilized is required.

Overall design flow:

In this in-class exercise, you will learn how to use the Time Division Multiplexor (TDM) by building a test bench for it and simulating. This will be directly applicable to creating your elevator controller in Lab4. This ICE will only be worth 50 pts (instead of 100) when averaged with the other ICEs.

Overview

The TDM allows you to rapidly switch between multiple data sets. This is especially useful when you only have one channel on which to send data (your seven segment display for instance). The TDM operation is fairly simple. It is driven by a clock and cycles between 4 different inputs continuously. Only one of the 4 inputs comes out of the TDM at a time. Lets start by going through each of the signals in the TDM entitiy. You can see a picture of the TDM entity in Figure 1 below.



```
entity TDM4 is
    generic ( constant k_WIDTH : natural  := 4); -- bits in input and output
    Port ( i_CLK         : in  STD_LOGIC;
           i_RESET       : in  STD_LOGIC; -- asynchronous
           i_D3          : in  STD_LOGIC_VECTOR (k_WIDTH - 1 downto 0);
           i_D2          : in  STD_LOGIC_VECTOR (k_WIDTH - 1 downto 0);
           i_D1          : in  STD_LOGIC_VECTOR (k_WIDTH - 1 downto 0);
           i_D0          : in  STD_LOGIC_VECTOR (k_WIDTH - 1 downto 0);
           o_DATA        : out STD_LOGIC_VECTOR (k_WIDTH - 1 downto 0);
           o_SEL         : out STD_LOGIC_VECTOR (3 downto 0)  -- selected data line (one-cold)
    );
end TDM4;
```

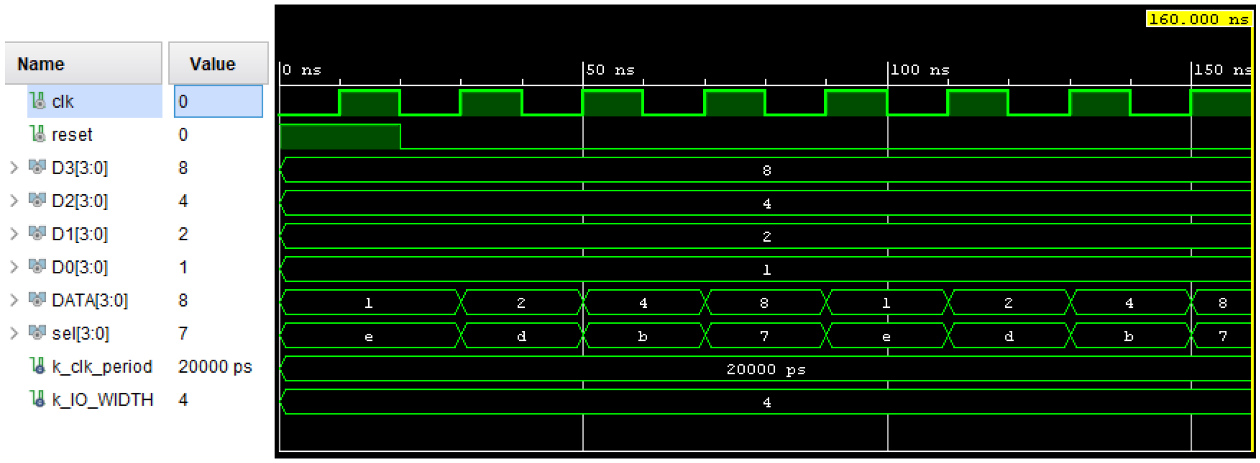
Figure 1. TDM Entity

Let's discuss each signal and what it does:

- K_WIDTH – is a generic constant. A constant is a set value used elsewhere (like our test bench clock period). The generic terminology means it is a value that can be set by the user when they instantiate the component. We will discuss later how it impacts this design
- i_CLK – is an input signal which drives the TDM. You will connect it to your test bench's simulated clk
- i_RESET – is an input signal which resets the TDM which sets the output to i_D0.
- i_D3 – is the first of 4 data sets which are passed into the TDM. Notice that it is a vector of k_WIDTH -1 downto 0). This means if K_WIDTH is 4 i_D3 is a 4 bit vector from 3 downto 0. K_WIDTH then allows the user to determine how big of a vector is going to be passed through. We will use 4 in this example, but if you were to use this with your seven segment decoder you could set it to 7.
- i_D2 – is the second of 4 data sets which are passed into the TDM.
- i_D1 – is the second of 4 data sets which are passed into the TDM.
- i_D0 – is the second of 4 data sets which are passed into the TDM.
- o_DATA – represents the output of the TDM. At any point in time it will be one of i_D3, i_D2, i_D1, or i_D0.
- o_SEL is an output vector that tells the user which of the input channels has been selected. It is one cold meaning "0111" indicates i_D3, "1011" indicates i_D2, "1101" indicates i_D1, and "1110" indicates i_D0. You will not need to use this signal in this exercise, other than to verify it is indeed working

Create a Testbench

1. Use the provided TDM4_tb file.
2. Finish the component declaration started in the test bench. Remember that you can grab this directly from the entity of the provided component.
3. Next we need to declare every signal we are going to use in this test bench. Essentially, every signal in our component's entity must have a companion signal in the test bench
   a. Declare the signal "i_CLK" of type std_logic
   b. Declare the constant "k_clk_period" of time time. Set it to 20 ns. This is our simulated clock.
   c. Declare the signal "i_RESET" of type std_logic
   d. Declare the constant k_IO_WIDTH of type natural and set it to 4.
   e. Declare the signals i_D3, i_D2, i_D1, i_D0, o_DATA of type std_logic_vector (k_IO_WIDTH – 1 downto 0).
   f. Declare the signal o_SEL of type std_logic_vector(3 downto 0).
4. Now complete the port map by connecting your newly declared signals to the signals in the TDM port map.
5. We need to finish setting up our clk process to drive our TDM. Finish the process using the same method as in our previous FSM test benches
6. The only thing we have left is to define our input vectors. Generally, when using the TDM these inputs will come from somewhere else (the output of your basic elevator controller in lab4 for instance). However, in this test we are going to hard code inputs. Assign the following values to the data inputs under the comment "—assign the values to data inputs"
   a. i_D3 should be "1100"
   b. i_D2 should be "1001"
   c. i_D1 should be "0110"
   d. i_D0 should be "0011"
7. Set your simulation to run for only 160 ns and run it. You should get a waveform that looks like Figure 2 below except your i_D3-i_D1 should be C, 9, 6 ,3 and the o_DATA should show 3 ,6 , 9, c, 3, 6, 9, c:



Name	Value	Waveform
clk	0	
reset	0	
D3[3:0]	8	8
D2[3:0]	4	4
D1[3:0]	2	2
D0[3:0]	1	1
DATA[3:0]	8	1, 2, 4, 8, 1, 2, 4, 8
sel[3:0]	7	e, d, b, 7, e, d, b, 7
k_clk_period	20000 ps	20000 ps
k_IO_WIDTH	4	4

Figure 2. Simulation Waveform

When you have finished, put a screenshot of your waveform and the answers to the following questions in a word document with your name and documentation statement and submit to your instructor.

1. **What does the TDM do?**
2. **What is the importance of k_WIDTH?**
3. **What is the purpose of o_SEL?**
4. **Why was your waveform different than the provided example (It's supposed to be)?**