FPGA course - Propulse and Orbit

- Martin Eggen

Contents

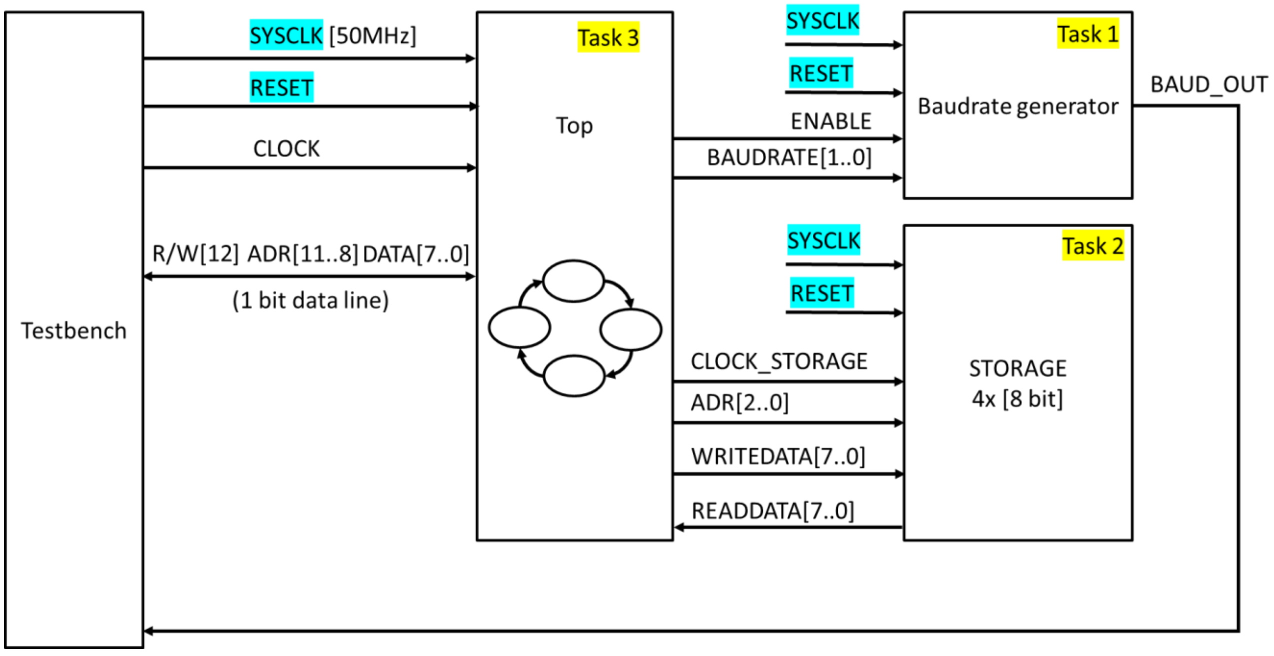
Intro

This course consists of three different exercises.

Task 1 – Baudrate generator with selectable baudrates

Task 2 – A four-byte storage

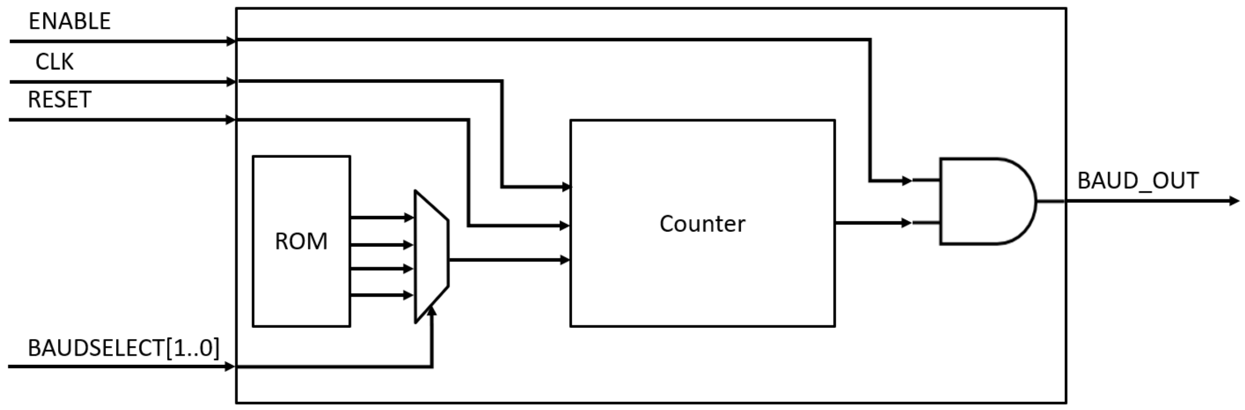
Task 3 – A component that connects to task 1 and 2, and acts like a slave that can communicate with a master using a custom protocol that you will be designing yourself.



Exercise 1 - Baudrate generator

In this exercise, you will be making a baudrate generator.

One solution could look something like this:



ENABLE - Active high enable signal that controls if the component should send out a signal or not.

SYSCLK - The system clock

RESET - Reset signal, active low. (RESET = ‘0’ → Reset component)

BAUDSELECT - in std_logic_vector(1 downto 0). Select which baudrate to use.

Start by making a new .vhd file in VS Code, and paste this in:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity baudgen is
  generic(
    g_clockspeed : integer := 50e6
  );
  port (
    CLK         : in std_logic;
    RST         : in std_logic;
    ENABLE      : in std_logic;
    BAUDSELECT : in std_logic_vector(1 downto 0);
    BAUD_OUT     : out std_logic := '0'
  );
end baudgen;

architecture rtl of baudgen is
  -- Signals
begin
  -- Process
end architecture;
```

The example code above includes all of the required ports for connecting the baudrate generator to a testbench and to other components.

Before you start, make sure you understand how a baudrate signal could be made:

We will be using our system clock for reference. The system clock will differ depending on what FPGA you are using. In this example, we will set the system clock speed using a *generic*.

To generate a specified baudrate, we need to change a signal from 0 to 1 and back to 0 every $\frac{system\ clock\ /\ baudrate}{2}$ system clock periods.

Example:

Desired baudrate is 4800 bits/s, and system clock is 50 MHz.

$$\frac{50 \cdot 10^6 / \ 4800}{2} = 5208.33 = 5209\ clock\ sycles$$

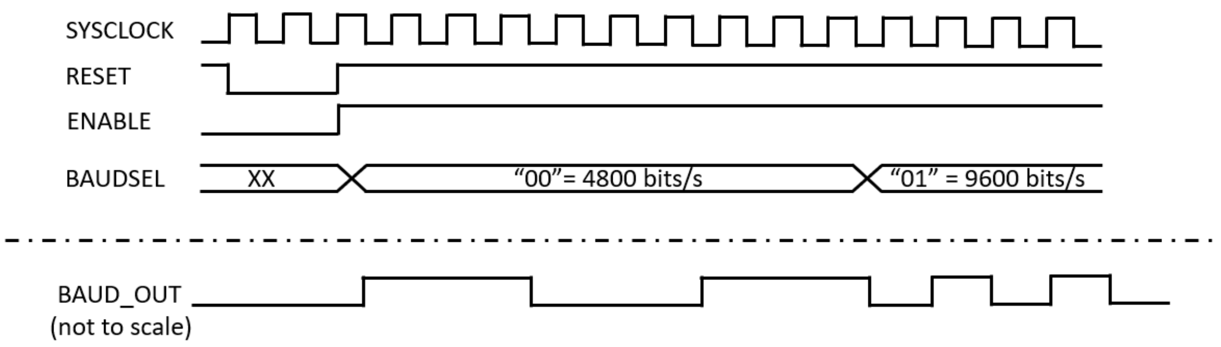
Make a read only memory (ROM) and some other signals you will need. The following code is just an example, and you can do it your own way if you like. Choose the same baudrates as used in the example if you don't want to edit the testbench. The signals should be located between *architecture* and *begin*

```
type ROM is array (0 to 3) of integer range 0 to g_clockspeed;
  constant baudPeriod : ROM := (g_clockspeed/4800, g_clockspeed/9600,
g_clockspeed/115_200, g_clockspeed/(g_clockspeed/2)); -- 4800, 9600, 115200, and
half of system clock

  signal s_counter        : integer range 0 to g_clockspeed/2;
  signal s_baudrate       : integer range 0 to g_clockspeed/2;
  signal s_output         : std_logic := '0';
  signal s_last_baudrate  : integer range 0 to g_clockspeed;
```

Begin by making a *process* that uses the system clock as an input. Then make it detect rising edge, and check if it *reset* is active or not.

```
process(CLK)
  begin
    if rising_edge(CLK) then
      if RST = '0' then
        -- Put your reset behaviour here
      else
        -- Put your code here
      end if;
    end if;
end process;
```

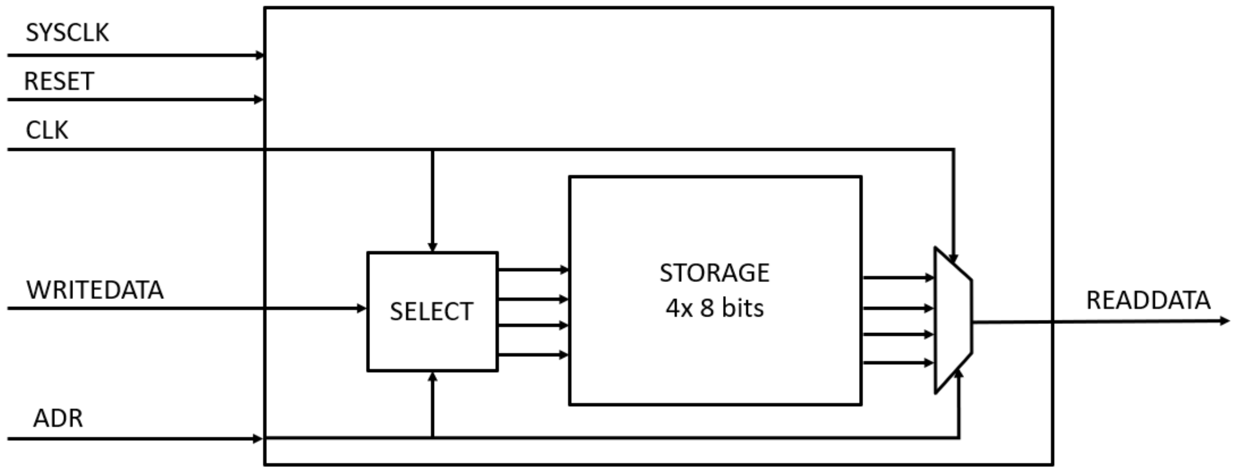
This is the communication protocol the testbench will be using towards your component.



Exercise 2 - Storage

In this exercise, you will me making a component that can store four bytes of data.

A solution could look something like this:



SYSCLK - The system clock

RESET - Reset signal, active low. (RESET = '0' → Reset component)

CLK - Another clock signal that goes high when the component should store or send data

WRITEDATA - in std_logic_vector(7 downto 0). Data written to the component.

READDATA - out std_logic_vector(7 downto 0). Data from the component.

ADR - Address. First bit indicates "Read not Write", meaning that the component should write data to storage when ADR(2) = '0', and send data over READDATA when ADR(2) = '1'.

ADR(1 downto 0) indicates which register it should be writing/reading to/from

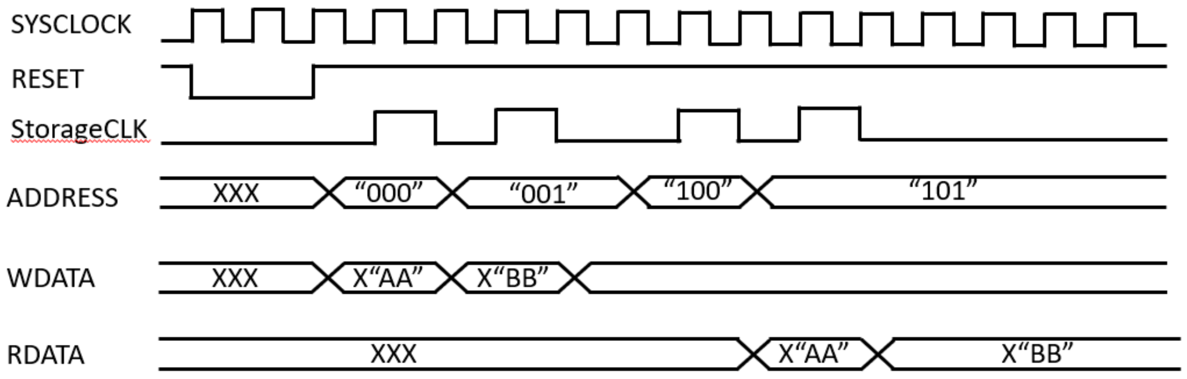
```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity storage is
  port (
    SYSCLK : in std_logic;
    RST : in std_logic;
    CLK : in std_logic;
    ADR : in std_logic_vector(2 downto 0);
    WRITEDATA   : in std_logic_vector(7 downto 0);
    READDATA    : out std_logic_vector(7 downto 0)
  );
end storage;

architecture rtl of storage is
  type t_STORAGE is array (0 to 3) of std_logic_vector(7 downto 0);
  signal STORAGE : t_STORAGE := (others => (others => '0'));

begin

  p_READ_nWRITE : process(SYSCLK) -- READ / n_WRITE
  begin
    if rising_edge(SYSCLK) then
      if RST = '0' then
        -- Reset behavior
      else
        -- Read and write behavior
      end if;
    end if;
  end process p_READ_nWRITE;
end architecture;
```

CASE could be used to solve these types of problems.

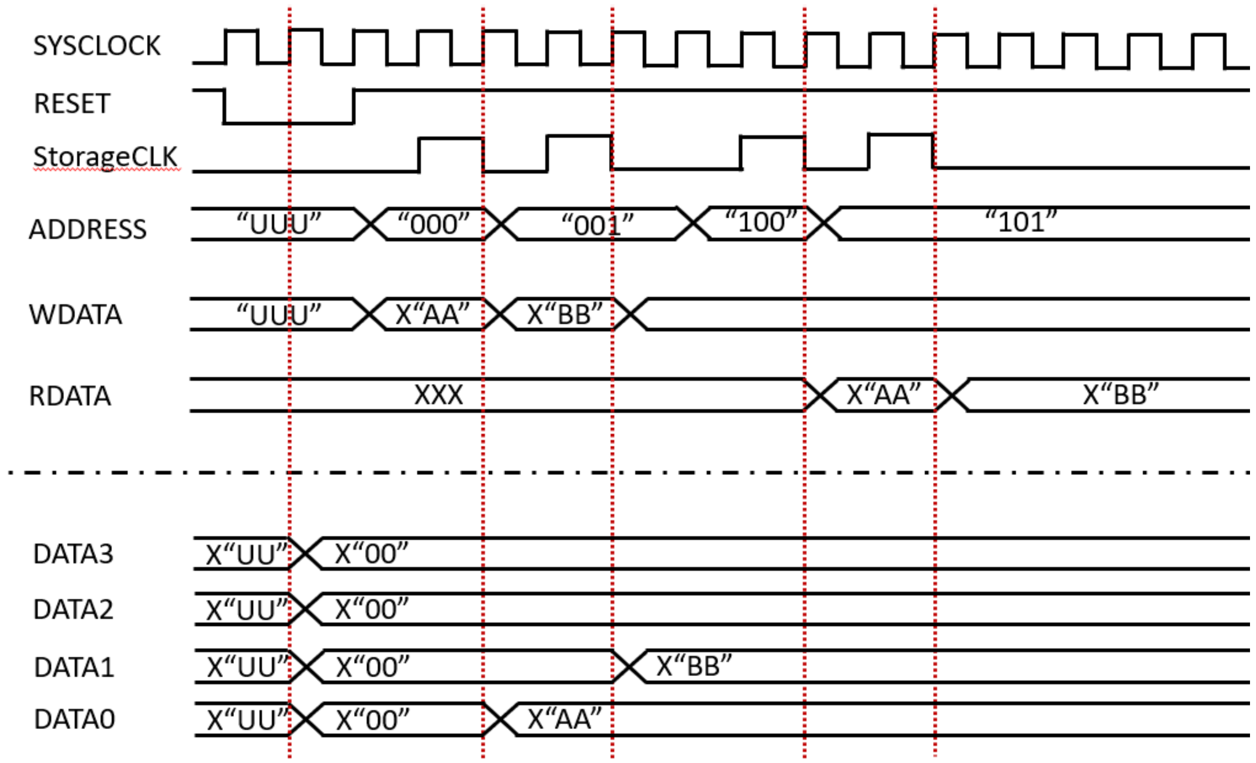
Here is an example you could use,

```
          case ADR(1 downto 0) is
            when "00" =>
            when "01" =>
            when "10" =>
            when "11" =>
            when others =>
                null;
            end case;
```

This is the communication protocol the testbench will be using towards your component.



With timing and internal storage:



Exercise 3 – Top level

The last exercise will be more open. This component is supposed to communicate with both the baudrate generator and the storage. (The provided testbenches and the wave diagrams may help to visualize how it should behave). The component should also be able to act as a slave in a bigger system. You will be designing the communication protocol between slave and master yourself. The only criteria are that it is serial communication, meaning that one and one bit is sent, and that it is controlled by an external clock. You can choose if it should be one wire for master to slave and another for slave to master, or if it should be a bidirectional line (a little bit more challenging).

You will also need to design an address map. An example is given below, but you can make your own. Make something that makes sense to you. This is for learning purpose, not efficiency or what's most common.

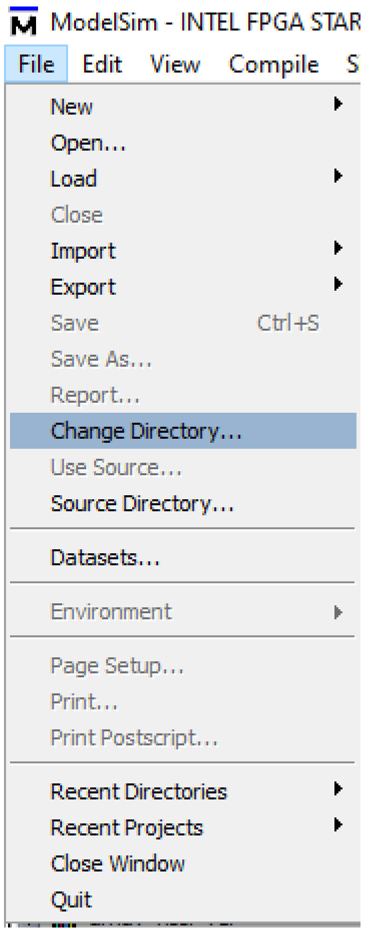
	12	11	10	9	8	7	6	5	4	3	2	1	0
Top	**R/nW**	**ADDRESS**				**DATA**							
Baudgen		**Select = "00"**									**BAUD EN**	**BAUDSELECT**	
Storage	**R/nW**	**Select = "01"**		**STORAGE ADDRESS**		**STORAGE DATA**							

Running a testbench

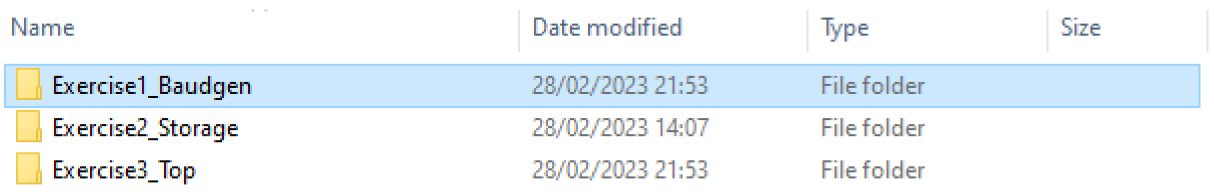
The testbenches will be provided, so all you have to do is to open Modelsim, compile and run the testbench.

Open Modelsim.

Go to **File-> Change Directory…**

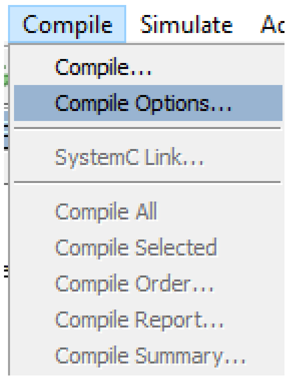


Select the folder than includes your code and testbench

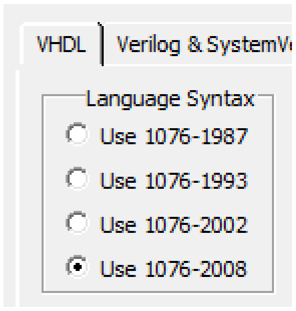


Name	Date modified	Type	Size
Exercise1_Baudgen	28/02/2023 21:53	File folder	
Exercise2_Storage	28/02/2023 14:07	File folder	
Exercise3_Top	28/02/2023 21:53	File folder	

Go to **Compile -> Compile Options...**



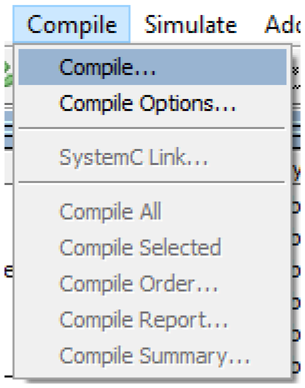
Select **2008**



Language Syntax

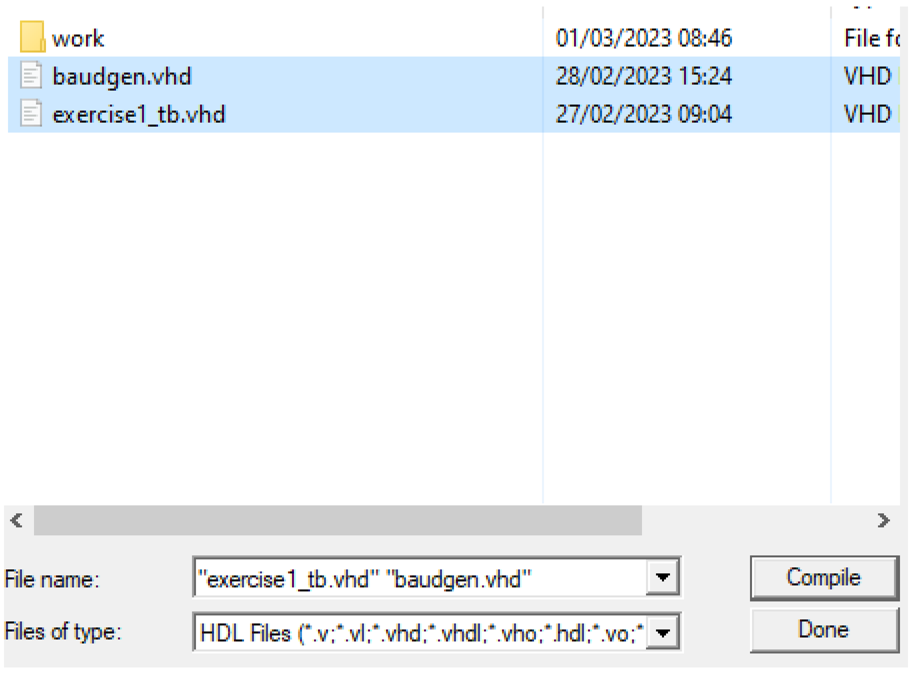
( ) Use 1076-1987
( ) Use 1076-1993
( ) Use 1076-2002
[✓ Use 1076-2008]

Then go to **Compile -> Compile...**



Compile the two files.

It will ask you if you want to create a “work”-folder the first time. Press *Yes / OK*



work	01/03/2023 08:46	File fo
baudgen.vhd	28/02/2023 15:24	VHD
exercise1_tb.vhd	27/02/2023 09:04	VHD

File name: "exercise1_tb.vhd" "baudgen.vhd"

Files of type: HDL Files (*.v;*.vl;*.vhd;*.vhdl;*.vho;*.hdl;*.vo;*

You will most likely end up with some errors.

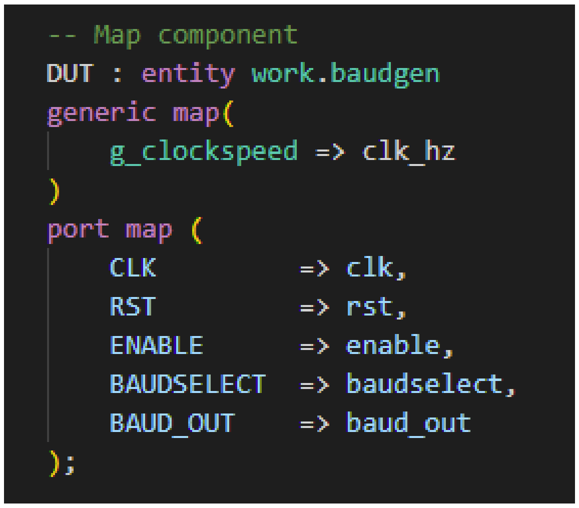
If the testbench has errors:

Make sure that the component you made matches with the testbench.

In the testbench code, locate where the component is mapped.

Check that the name *work.[your component]*, and all of the ports matches.

(Your ports are the one on the left)

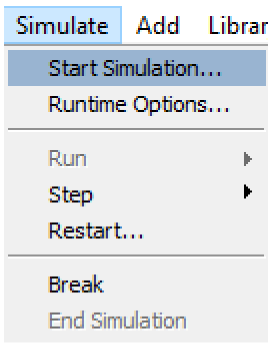


```
-- Map component
DUT : entity work.baudgen
generic map(
    g_clockspeed => clk_hz
)
port map (
    CLK         => clk,
    RST         => rst,
    ENABLE      => enable,
    BAUDSELECT  => baudselect,
    BAUD_OUT    => baud_out
);
```

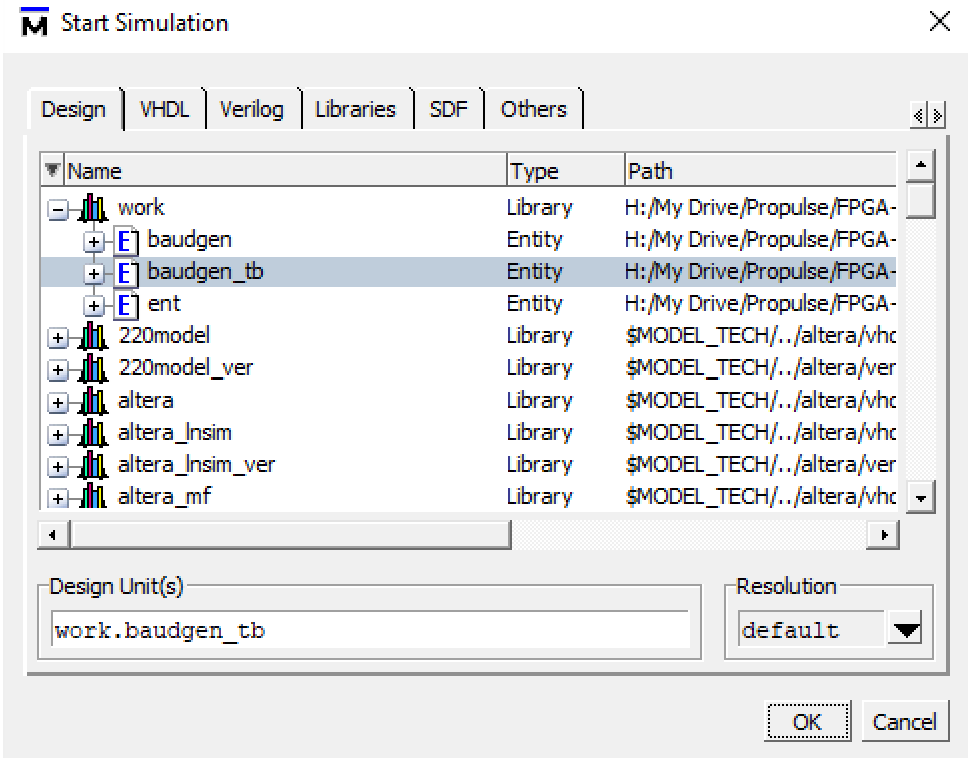
Simulate

After you have resolved all of your errors, you are ready to run a simulation.

Simulate -> Start Simulation...



Simulate the corresponding testbench



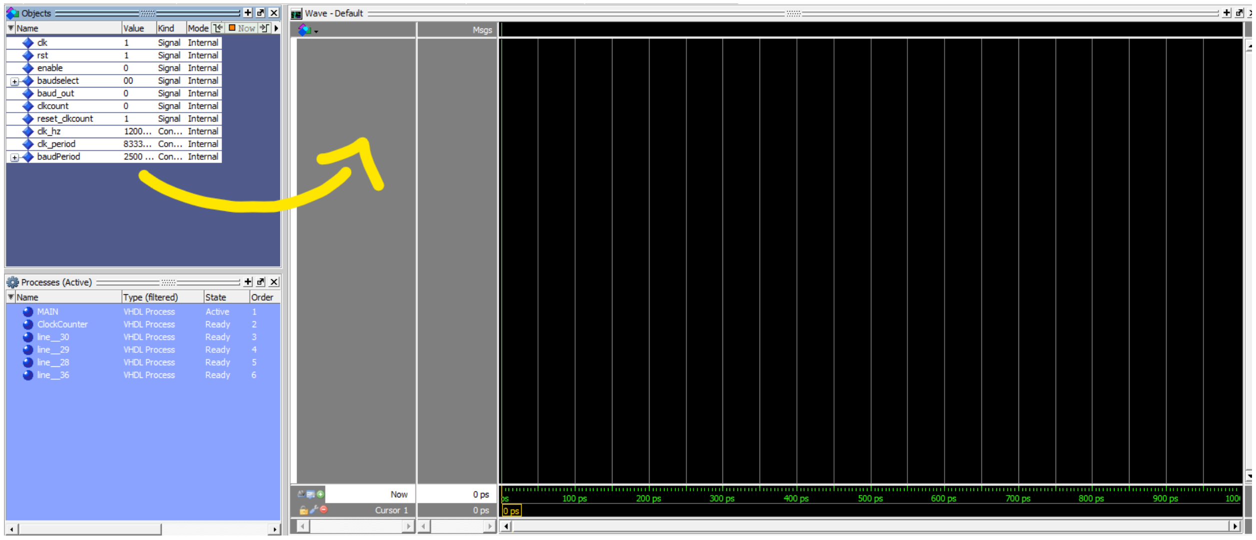
Name	Type	Path
work	Library	H:/My Drive/Propulse/FPGA-
baudgen	Entity	H:/My Drive/Propulse/FPGA-
baudgen_tb	Entity	H:/My Drive/Propulse/FPGA-
ent	Entity	H:/My Drive/Propulse/FPGA-
220model	Library	$MODEL_TECH/../altera/vhc
220model_ver	Library	$MODEL_TECH/../altera/ver
altera	Library	$MODEL_TECH/../altera/vhc
altera_lnsim	Library	$MODEL_TECH/../altera/vhc
altera_lnsim_ver	Library	$MODEL_TECH/../altera/ver
altera_mf	Library	$MODEL_TECH/../altera/vhc

Design Unit(s): `work.baudgen_tb`

Resolution: `default`

This will open a new window called *Wave*. If this does not happen automatically, go to **View -> Wave**

Drag and drop all of the signals into the Wave window.



Objects

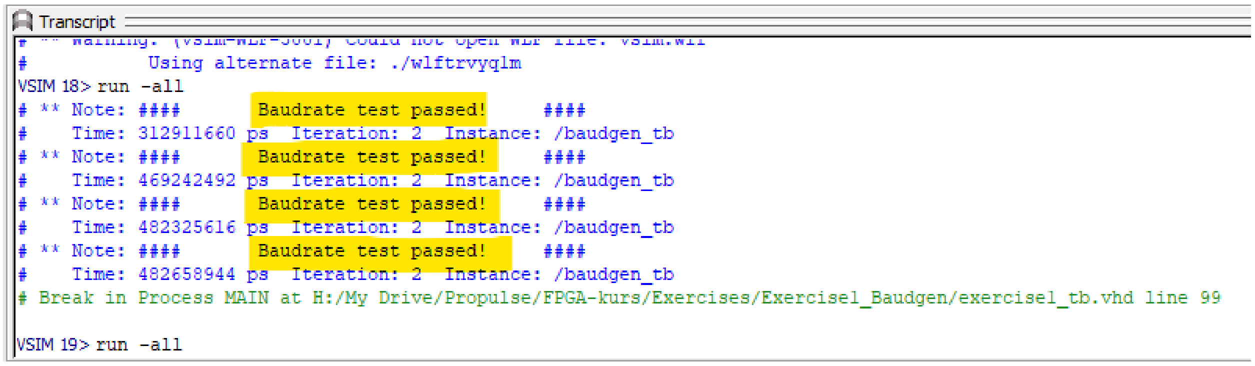
Name	Value	Kind	Mode
clk	1	Signal	Internal
rst	1	Signal	Internal
enable	0	Signal	Internal
baudselect	00	Signal	Internal
baud_out	0	Signal	Internal
clkcount	0	Signal	Internal
reset_clkcount	1	Signal	Internal
clk_hz	1200…	Con…	Internal
clk_period	8333…	Con…	Internal
baudPeriod	2500 …	Con…	Internal

Processes (Active)

Name	Type (filtered)	State	Order
MAIN	VHDL Process	Active	1
ClockCounter	VHDL Process	Ready	2
line__30	VHDL Process	Ready	3
line__29	VHDL Process	Ready	4
line__28	VHDL Process	Ready	5
line__36	VHDL Process	Ready	6

Press **CTRL+S** and save the waveform.

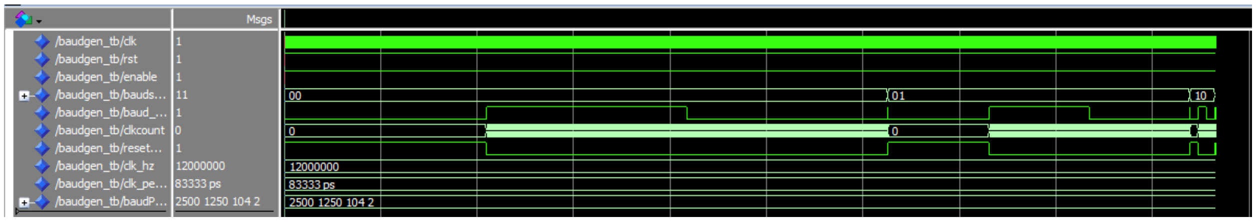
Type **run -all** in the terminal or navigate to **Simulate -> Run -> Run -All**



Transcript

```
# ** Warning: (vsim-WLF-5001) Could not open WLF file: vsim.wlf
#           Using alternate file: ./wlftrvyqlm
VSIM 18> run -all
# ** Note: ####   Baudrate test passed!     ####
#    Time: 312911660 ps  Iteration: 2  Instance: /baudgen_tb
# ** Note: ####   Baudrate test passed!     ####
#    Time: 469242492 ps  Iteration: 2  Instance: /baudgen_tb
# ** Note: ####   Baudrate test passed!     ####
#    Time: 482325616 ps  Iteration: 2  Instance: /baudgen_tb
# ** Note: ####   Baudrate test passed!     ####
#    Time: 482658944 ps  Iteration: 2  Instance: /baudgen_tb
# Break in Process MAIN at H:/My Drive/Propulse/FPGA-kurs/Exercises/Exercise1_Baudgen/exercise1_tb.vhd line 99

VSIM 19> run -all
```

Press “F” to Zoom Fit. Your wave diagram should look something like this.



	Msgs
/baudgen_tb/clk	1
/baudgen_tb/rst	1
/baudgen_tb/enable	1
/baudgen_tb/bauds…	11
/baudgen_tb/baud_…	1
/baudgen_tb/clkcount	0
/baudgen_tb/reset…	1
/baudgen_tb/clk_hz	12000000
/baudgen_tb/clk_pe…	83333 ps
/baudgen_tb/baudP…	2500 1250 104 2

Do-file

You are now ready to run the DO-file every time you want to run the simulation again. This is optional, but will save some time.

```
quit -sim

vcom baudgen.vhd
vcom exercise1_tb.vhd

vsim baudgen_tb
do wave.do
run -all
wave zoom full
```

A DO-file is just running commands through the terminal in Modelsim automatically.

Make sure that the names matches and that the testbench is the last one to be compiled.

Run the DO-file by typing ***do sim.do*** in the terminal.