CE1911 LABORATORY PROJECT

SUMMARY

Memory is the ability to retain information indefinitely. In humans, memory retains patterns experienced by our bodies through its sensors. These memories include reflexive responses learned from experiences such as touching hot surfaces, as well as behaviors and information learned by repetition. Human memory is a complex system of cell interconnections controlled by neurotransmitters. Reinforcement of stored information strengthens the biological memory of it.

Computer memory retains voltage patterns representing binary numbers. Computer programmers and computer architects attach meaning to these numbers and use them to direct calculation and computer behavior. For example, a computer program written in C or Swift is a sequence of control and calculation steps that occur through time. A compiler converts this sequence to binary numbers stored in program memory. The computer executes each instruction by reading it from program memory, deciphering it into control signals for the functional logic elements of the calculation engine, and storing any resulting data value from the instruction back to memory.

Computer architects have developed different types of computer memories. Each type works to optimize one or more of these areas: speed, size, power, and cost.

Static memory does not lose its voltages as long as the systems supplies battery power to the memory circuit. This adjective “static” describes this stability in memory. Retention of the data through time occurs because the output of the memory feeds back to become its input again. This constantly reinforces the retained value and makes a stable memory. The reinforcement circuit requires multiple gates and thus multiple transistors. Because this type of memory does not forget, it does not need to be periodically refreshed to ensure data integrity. This optimizes speed because the system does not need to take any units of time to refresh forgotten data.

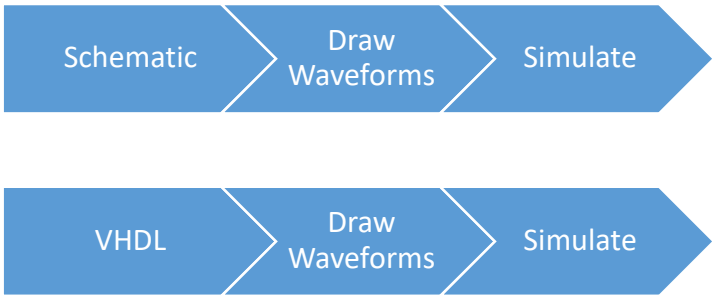
Dynamic memory, on the other hand, optimizes size and thus provides larger memory circuits. Multiple gates do not provide reinforcement of the remembered value. Instead, this type of memory stores logic-1 as electrical charge on a single capacitor. It does not use multiple gates and thus uses minimal transistors per bit. The capacitor immediately begins decaying the stored voltage toward logic-0 because of parasitic resistive-capacitive effects caused in built material. The RC time constant determines the decay rate. Electric circuit theory courses will cover the principles of RC time constants and capacitive decay. This type of memory forgets stored logic-1s because the capacitor parasitically decays logic-1 to logic-0. Thus, a refresh controller must periodically refresh every bit of memory. This controller is another piece of hardware that reads each memory bit and rewrites it. The controller must move through all the memory bits before any bit forgets its stored logic-1. Thus, there is an operational speed disadvantage. A computer program attempting to access memory may have to wait for the refresh controller to finish working on the current set of bits before the memory system grants access to store data into memory.

These laboratory exercises focus on static memories. You will build and simulate a schematic circuit for a 1-bit D flip-flop with synchronous load and synchronous reset. You will then use the DFF component to build and simulate a schematic circuit for a byte-sized register.

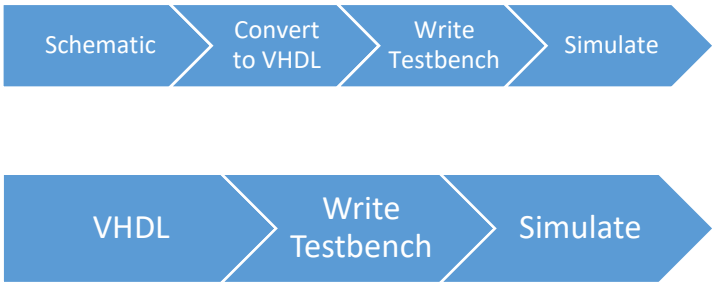
CE1911 LABORATORY PROJECT

NEW SIMULATION WORKFLOW

Simulation tools are an important part of modern engineering design workflow. These tools help to ensure that designs meet functional and timing requirements. Computer-aided design suites often include multiple ways to complete simulation. One technique uses input waveforms drawn by hand with a waveform editor. CE1901 used this approach. A second technique uses a hardware description language to describe the input voltage changes through time. Engineers call these hardware descriptions of test inputs **test benches**. CE1911 will use this approach.



CE1901 ModelSim-Altera Simulation Workflow



CE1911 ModelSim-Altera Simulation Workflow

ModelSim requires that engineers convert all designs to textual hardware descriptions before simulation. The Quartus waveform editor completes this process automatically when it starts a simulation. Engineers must complete this step when using VHDL testbenches.

CE1911 LABORATORY PROJECT

DELIVERABLES DUE AT THE END OF THE LABORATORY PERIOD

1. **Demonstrate** a working Quartus DFF schematic project and ModelSim simulation to the instructor.
2. **Demonstrate** a working Quartus REG8 schematic project and ModelSim simulation to the instructor.
3. **Submit** a laboratory report using the preferred method of your instructor that includes these sections:
    a. **Abstract**: no more than one page describing the circuits and the work you have completed. This section should use text to convey to the reader the foundational principles of the memory circuits, outline the steps you took to complete the project, and summarize your results.
    b. **Circuits**: a section that includes the Quartus schematic diagram, Quartus generated RTL diagram, and any text you wish to include that demonstrates your understanding of the work.
    c. **Simulations**: a section that documents the method of testing for each circuit. In other words, given the inputs, how did you choose the number line you applied as voltages? Did it follow a truth table? Was it random? Did you set arbitrary values? Why did you choose your method? You must provide the ModelSim simulation waveforms for each circuit with hand-written, electronic pen written, or typed text that demonstrates how you know the simulation verifies the circuit operates correctly.
    d. **Conclusion**: A short section that describes how the laboratory reinforced your learning of the course material. Include any questions you still have so that the instructor can provide additional help if needed.

DFF1911 LABORATORY PROJECT

1. **Create** a Quartus schematic blueprint project called **dff1911**.
2. **Implement** a gate-level DFF block diagram with synchronous load and synchronous clear.
    a. **Use** Reference Figures 1-4 at the end of this document to guide your work.
    b. **Complete** Analysis and Synthesis on top-level design **dff1911** to ensure there are no errors.
    c. **Correct** any errors and **recompile** if needed.
3. **Convert** each block diagram file to a **VHDL** version using

    File → Create/Update → Create HDL Design File from Current File

4. **Add** the VHDL files to the project and **remove** the block diagram files.

    Project → Add/Remove Files In Project…
    Select BDF and Remove
    Browse and add VHDL text files

5. **Set** dff1911.vhd as the top-level entity.

    Open the de1911.vhd file as the active tab.
    Project → Set as Top Level Entity

6. **Add** a new VHDL file to the project. **Enter** the text in Figure 5 and **save** as **testbench_dff1911.vhd**.
7. **Rebuild** the design using **Start Analysis and Synthesis**.

8. **Configure** the simulation tool.

Assignments → Settings → EDA Tool Settings → Simulation	
Tool Name	**ModelSim-Altera**
Format for Output Netlist	**VHDL**
NativeLink Settings	**Compile Test Bench**
Test Benches…	
New…	
Test Bench Name (must match test bench entity)	**testbench_dff1911**
Top Level Module in Test Bench	**testbench_dff1911**
End Simulation At (determined by VHDL time delays)	**6 us**
Filename (browse, select, add)	**testbench_dff1911.vhd**
Tools → Options → EDA Tool Options	
Model-Sim Altera	**C:\intelFPGA_lite\18.0\modelsim_ase\win32aloem**

9. **Tools → Run Simulation Tool RTL Simulation**
   a. **ModelSim** will open and run the VHDL test bench to control the simulation. ModelSim will likely remain behind Quartus. Select the ModelSim icon from the Windows task bar to bring it to the front.
   b. **Verify** the circuit operates correctly. ModelSim will draw the waveforms in the output window. **Click** in the waveforms. **Use** keyboard keys **I** and **O** to zoom **in** and **out**.

REG4 LABORATORY PROJECT

1. **Create** a Quartus schematic blueprint project called **reg4**.
2. **Implement** a functional-level REG4 block diagram. **Use** Reference Figure 6 as a guide.
3. **Note** that the register interconnects four dff1911 flip-flops. Thus, you must copy the blueprints for the SR latch, D latch, and dff1911 flip-flop into the new project and create symbols files for each.
4. **Complete** Analysis and Sythesis on top-level design REG4 to ensure you have no errors. **Correct** if needed.
5. **Convert** the schematics to VHDL files.
6. **Remove** the schematics and **add** the VHDL files.
7. **Set** reg4.vhd as the top-level entity and **rebuild** the design.
8. **Add** a new VHDL file and enter the test bench from Figure 7. **Save** the file as **testbench_reg4.vhd**.
9. **Configure** the simulator, **run** functional simulation, and **verify** correct operation.

Assignments → Settings → EDA Tool Settings → Simulation	
Tool Name	**ModelSim-Altera**
Format for Output Netlist	**VHDL**
NativeLink Settings	**Compile Test Bench**
Test Benches…	
New…	
Test Bench Name (must match test bench entity)	**testbench_reg4**
Top Level Module in Test Bench	**testbench_reg4**
End Simulation At (determined by VHDL time delays)	**800 ns**
Filename (browse, select, add)	**testbench_reg4.vhd**

CE1911 LABORATORY PROJECT

REFERENCE FIGURES FOR DFF1911 LABORATORY PROJECT

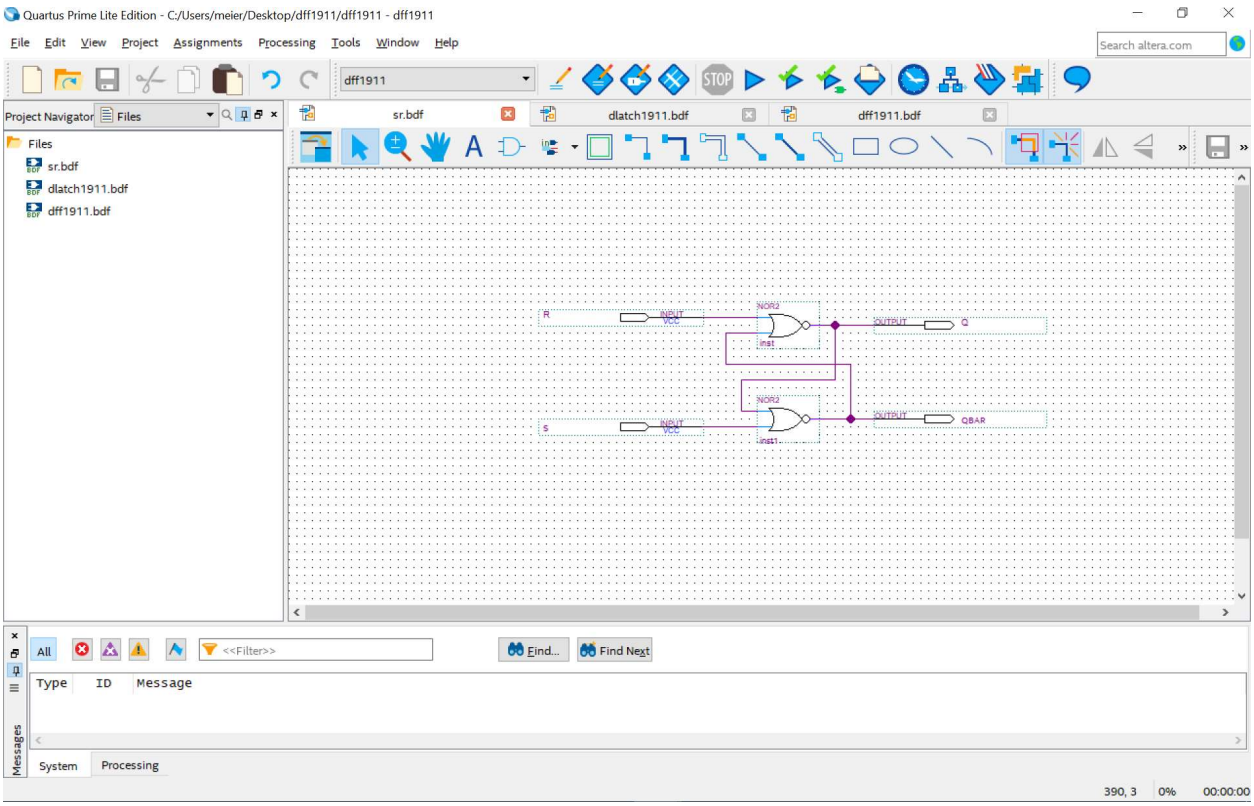


Figure 1: SR Latch with Active-High Control Signals

CE1911 LABORATORY PROJECT

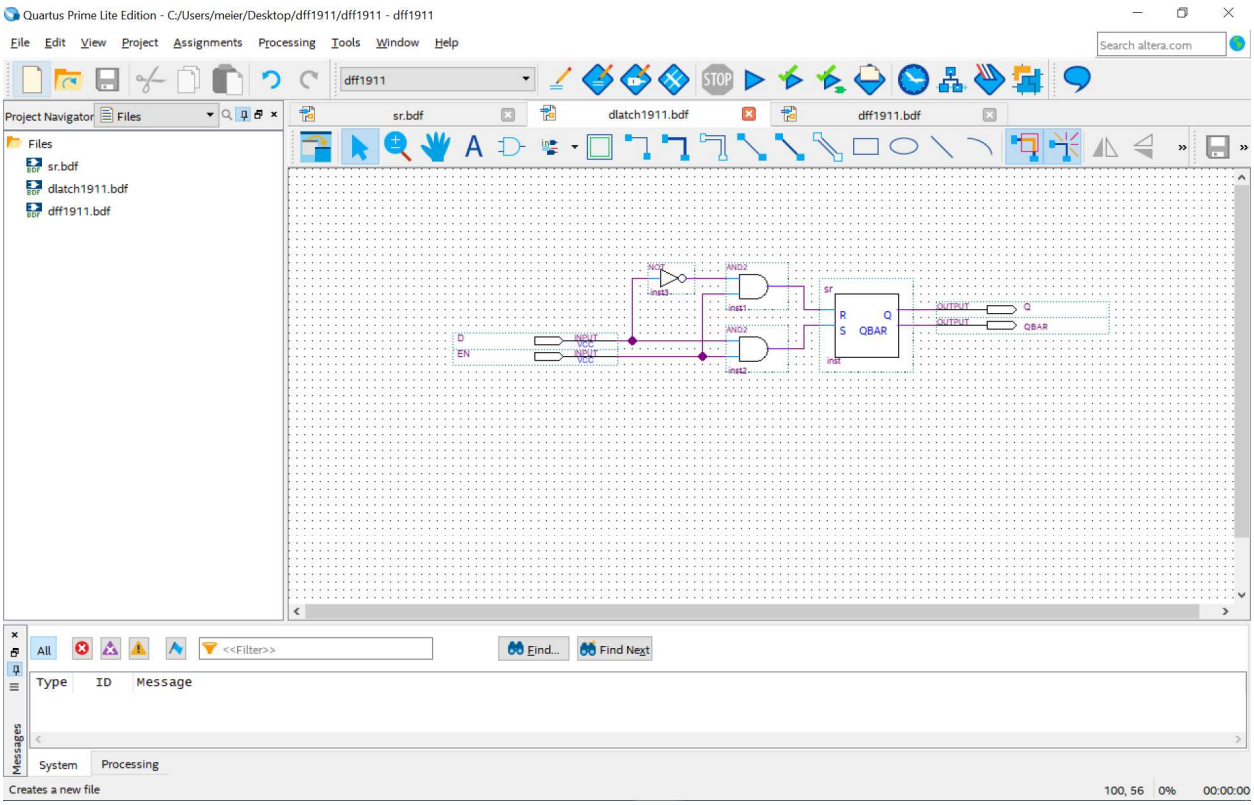


Figure 2: D Latch with Enable

INPUTS						
CLR	LD	D	Q	DESIRED BEHAVIOR		DINT
0	0	0	0	hold Q		0
0	0	0	1	hold Q		1
0	0	1	0	hold Q		0
0	0	1	1	hold Q		1
0	1	0	0	load and store d		0
0	1	0	1	load and store d		0
0	1	1	0	load and store d		1
0	1	1	1	load and store d		1
1	0	0	0	clear Q to 0		0
1	0	0	1	clear Q to 0		0
1	0	1	0	clear Q to 0		0
1	0	1	1	clear Q to 0		0
1	1	0	0	clear Q to 0		0
1	1	0	1	clear Q to 0		0
1	1	1	0	clear Q to 0		0
1	1	1	1	clear Q to 0		0

Figure 3: Truth Table Used to Design the DFF1911 Circuit

- The DFF1911 component has active-high **synchronous load** and **synchronous clear** signals.
- Synchronous control signals cause behavior at the clock-edge. They are **edge-triggered behaviors**.
- Designers sometimes rename load as **enable**. The name load seems to be more common.
- Designers sometimes rename clear as **reset** because it better matches the foundational SR latch signal.
- The CLR behavior has a higher priority than the load (enable) signal in the design. (**CLR overrides LD**)
- The truth table highlights the active-high behaviors in blue.
- Sequential logic circuits feed the memory output, Q, back into the circuit as an input. This allows the current remembered value to help calculate the next remembered value.
- Boolean algebra or the K-map technique result in this equation:

$$DINT = \overline{CLR} \cdot \overline{LD} \cdot Q + \overline{CLR} \cdot LD \cdot D$$

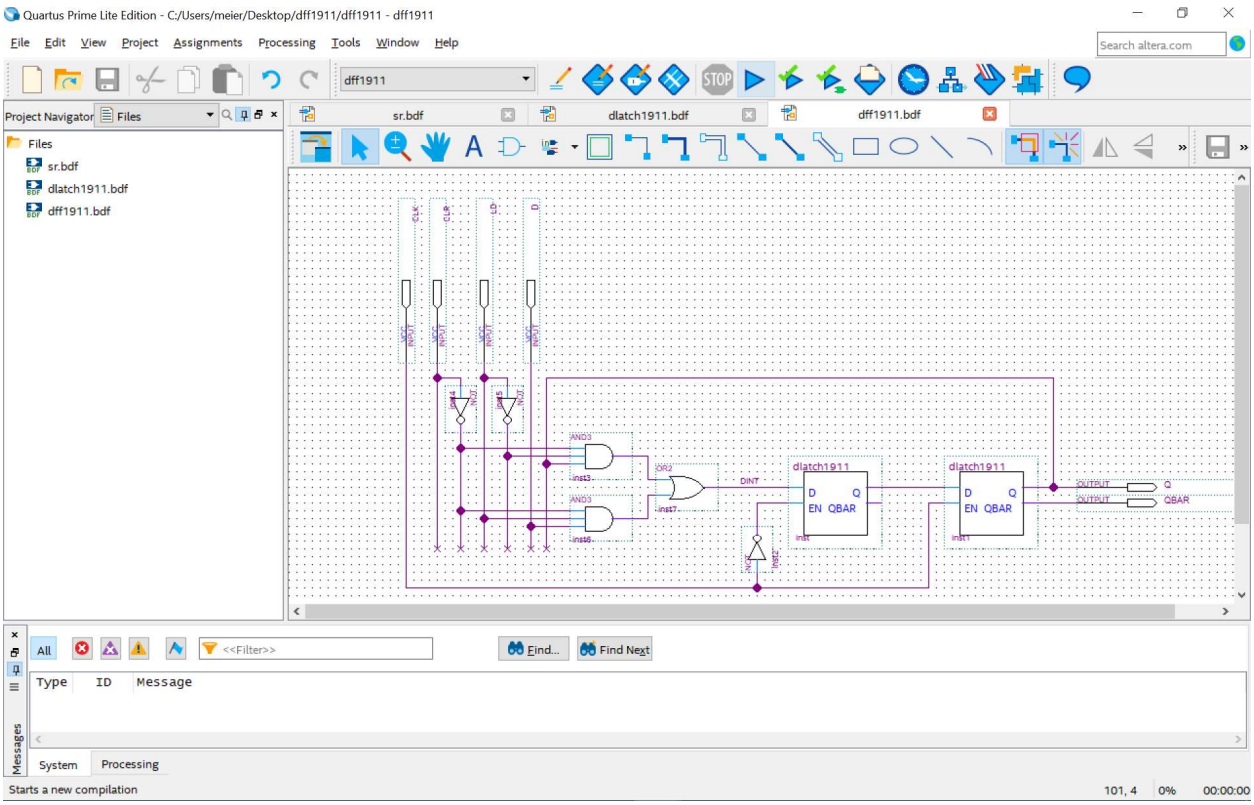


Figure 4: DFF with Active-High Clear (Reset) and Active-High Load (Enable)

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

entity testbench_dff1911 is
end entity testbench_dff1911;

architecture dataflow of testbench_dff1911 is
  signal CLK, CLR, LD, D: std_logic;
  signal Q, QBAR: std_logic;

  -- Ensure this component statement matches the dff1911.vhd entity statement
  -- by adjustin if needed.
  component dff1911 is
  port(CLR,LD,D,CLK: in std_logic;
       Q,QBAR: out std_logic);
  end component dff1911;

begin

  -- place the unit under test
  UUT: dff1911 port map(CLR=>CLR,LD=>LD,D=>D,CLK=>CLK,Q=>Q,QBAR=>QBAR);

  -- write the systematic process of testing: time delayed voltage changes
  test: process
  begin
          CLK <= '0'; CLR <= '0'; LD <= '0'; D <= '0'; WAIT FOR 0.5US;
          CLK <= '1'; CLR <= '0'; LD <= '0'; D <= '0'; WAIT FOR 0.5US;
          CLK <= '0'; CLR <= '0'; LD <= '0'; D <= '1'; WAIT FOR 0.5US;
          CLK <= '1'; CLR <= '0'; LD <= '0'; D <= '1'; WAIT FOR 0.5US;
          CLK <= '0'; CLR <= '0'; LD <= '1'; D <= '0'; WAIT FOR 0.5US;
          CLK <= '1'; CLR <= '0'; LD <= '1'; D <= '0'; WAIT FOR 0.5US;
          CLK <= '0'; CLR <= '0'; LD <= '1'; D <= '1'; WAIT FOR 0.5US;
          CLK <= '1'; CLR <= '0'; LD <= '1'; D <= '1'; WAIT FOR 0.5US;
          CLK <= '0'; CLR <= '0'; LD <= '0'; D <= '1'; WAIT FOR 0.5US;
          CLK <= '1'; CLR <= '0'; LD <= '0'; D <= '1'; WAIT FOR 0.5US;
          CLK <= '0'; CLR <= '1'; LD <= '0'; D <= '1'; WAIT FOR 0.5US;
          CLK <= '1'; CLR <= '1'; LD <= '0'; D <= '1'; WAIT FOR 0.5US;
          CLK <= '1'; CLR <= '0'; LD <= '0'; D <= '1'; WAIT FOR 0.5US;
  end process test;

 end architecture dataflow;
```

Figure 5: VHDL Test Bench for DFF1911

CE1911 LABORATORY PROJECT

REFERENCE FIGURES FOR REG4 LABORATORY PROJECT

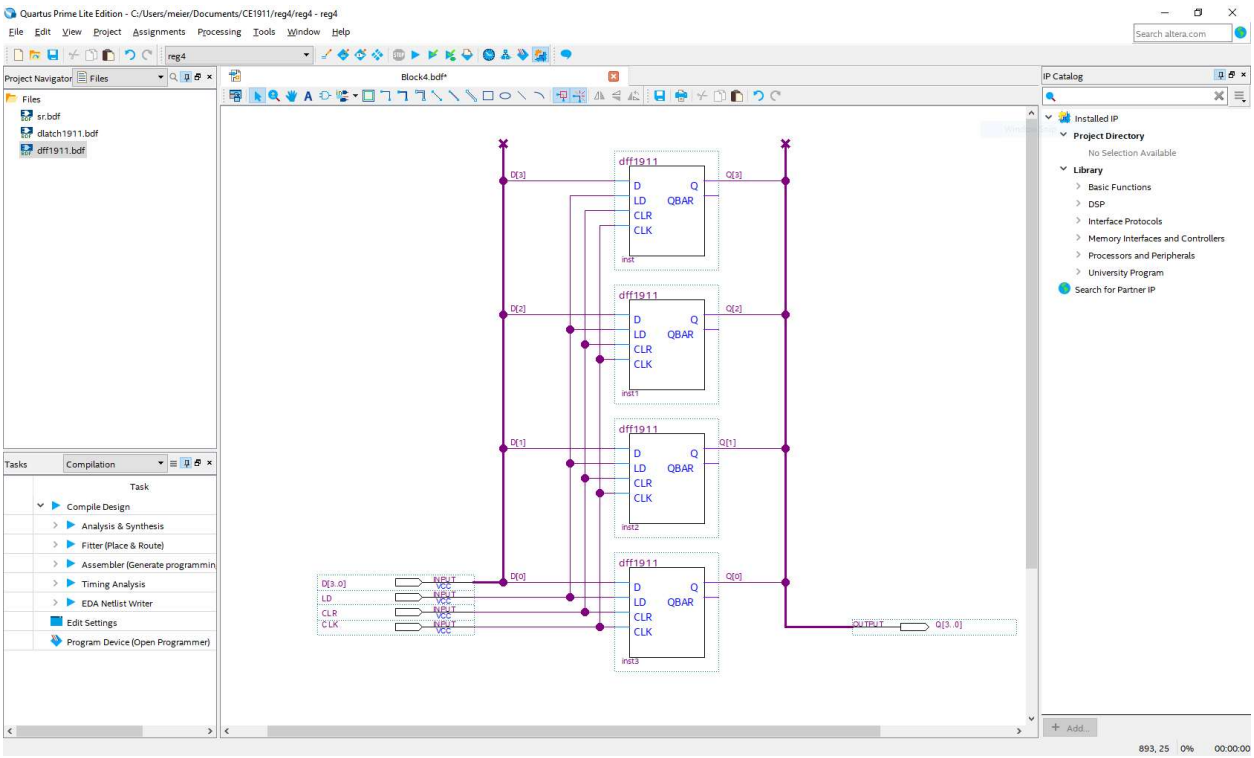


Figure 6: 4-bit Register with Synchronous Load and Synchronous Clear

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

entity testbench_reg4 is
end entity testbench_reg4;

architecture dataflow of testbench_reg4 is

  signal CLK, LD, CLR: std_logic; -- control signals
  signal D: std_logic_vector(3 downto 0); -- data inputs
  signal Q: std_logic_vector(3 downto 0); -- memory output

  -- ensure this component statement matches the reg4.vhd entity statement
  -- by adjusting if needed.
  component reg4 is
  port
    (CLK: in std_logic;
     LD:  in std_logic;
     CLR: in std_logic;
     D:   in std_logic_vector(3 DOWNTO 0);
     Q:   out std_logic_vector(3 DOWNTO 0));
  end component reg4;

begin

  -- place the unit under test
  UUT: reg4 port map(CLK=>CLK, LD=>LD, CLR=>CLR, D=>D, Q=>Q);

  -- write the systematic process of testing: time delayed voltage changes
  test: process
  begin
          CLK <= '0'; CLR <= '0'; LD <= '0'; D <= X"4"; WAIT FOR 50ns;
          CLK <= '1'; CLR <= '0'; LD <= '0'; D <= X"5"; WAIT FOR 50ns;
          CLK <= '0'; CLR <= '0'; LD <= '0'; D <= X"5"; WAIT FOR 50ns;
          CLK <= '1'; CLR <= '0'; LD <= '0'; D <= X"5"; WAIT FOR 50ns;
          CLK <= '0'; CLR <= '0'; LD <= '1'; D <= X"5"; WAIT FOR 50ns;
          CLK <= '1'; CLR <= '0'; LD <= '1'; D <= X"5"; WAIT FOR 50ns;
          CLK <= '0'; CLR <= '0'; LD <= '1'; D <= X"A"; WAIT FOR 50ns;
          CLK <= '1'; CLR <= '0'; LD <= '1'; D <= X"A"; WAIT FOR 50ns;
          CLK <= '0'; CLR <= '0'; LD <= '0'; D <= X"C"; WAIT FOR 50ns;
          CLK <= '1'; CLR <= '0'; LD <= '0'; D <= X"C"; WAIT FOR 50ns;
          CLK <= '0'; CLR <= '1'; LD <= '0'; D <= X"C"; WAIT FOR 50ns;
          CLK <= '1'; CLR <= '1'; LD <= '0'; D <= X"C"; WAIT FOR 50ns;
          CLK <= '1'; CLR <= '0'; LD <= '0'; D <= X"C"; WAIT FOR 50ns;

  end process test;

end architecture dataflow;
```

Figure 7: VHDL Test Bench for REG4