ECEN 468 Advanced Logic Design
Department of Electrical and Computer Engineering
Texas A&M University

(Lab exercise created by Jaeyeon Won and Jiang Hu)

Lab 8

Design of UART Transmitter

Purpose:

In this lab, we will design a transmitter of Universal Asynchronous Receiver/Transmitter (UART) with Verilog, and will use Design Analyzer as a tool for synthesis. This module will be attached to our entire system later, so can transmit data to other devices or processors.

Preparation:

1. Brief introduction to UART.

The UART is a type of "asynchronous receiver/transmitter", a piece of computer hardware that translates data between parallel and serial forms. UARTs are commonly used in conjunction with communication standards such as EIA RS-232, RS-422 or RS-485. The universal designation indicates that the data format and transmission speeds are configurable. Figure 1 shows general description of communication between processors through a serial channel. Those processors communicate internally with parallel data to speed up and use a serial channel to communication with other processors to reduce the number of wires, so decrease the cost of hardware.

Figure 1 Processor/modem communication over a serial channel

For this lab, a UART transmits 8-bits data and augmented by a parity bit that can be used for error detection. For transmission, the modem wraps this 9-bit sub-word with a start-bit in the least significant bit (LSB), and a stop-bit in the most significant bit (MSB), resulting in the 10-bit word format shown in Figure 2. The first 9 data bits of the word are transmitted in sequence, beginning with the start-bit, with each bit being asserted at the serial line for one cycle (bit-time) of the modem clock. The stop-bit may assert for more than one clock.

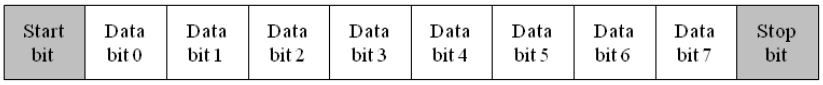


Figure 2 Data format for ASCII text transmitted by a UART

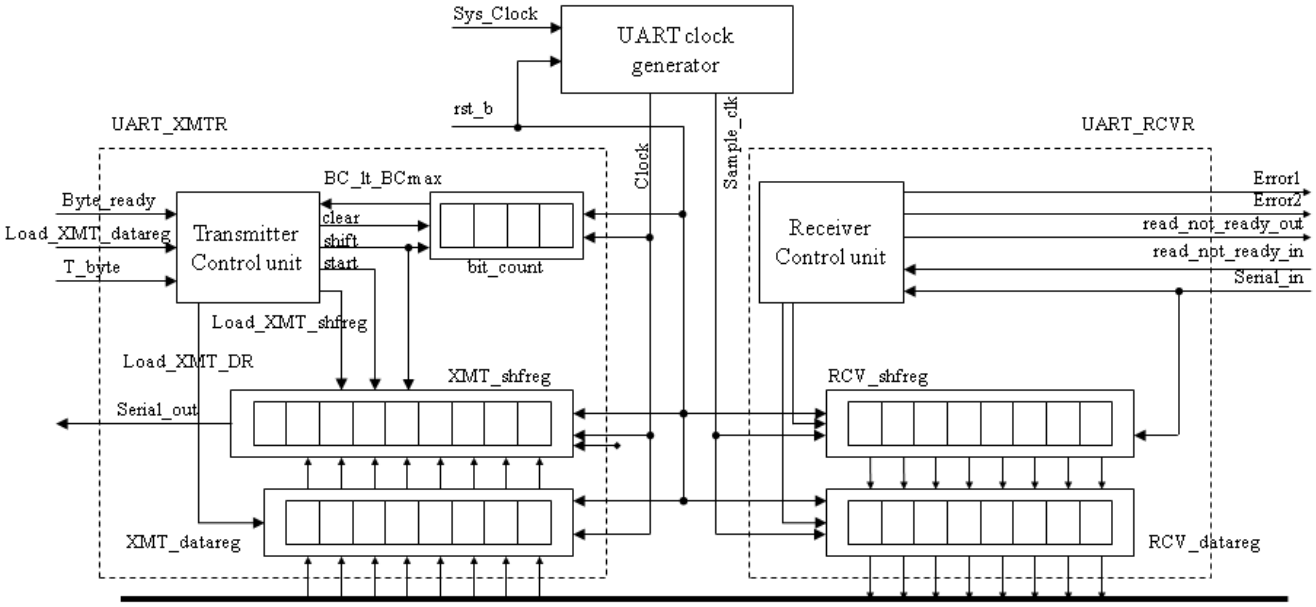


Figure 3 Block diagram of a UART

The simplified architecture of a UART presented in Figure 3. It shows the signals used by a host processor to control the UART and to move data to and from a data bus in the host machine.

Input: Load_XMT_datareg, Byte_ready, T_byte, rst_b (1 bit), Data_Bus (8 bits)
Output : Serial_out (1 bit)

Local Storage : XMT_datareg (8 bits), XMT_shftreg (9 bits), bit_count (4 bits)

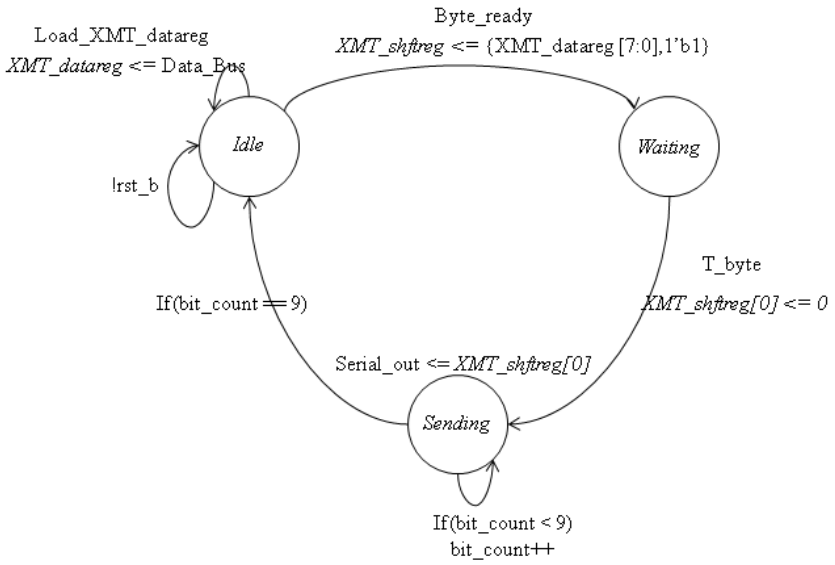


Figure 4 High-Level State Machine (HLSM) of the transmitter

The input-output signals of the transmitter are shown in the high-level block diagram in Figure 5. The input signals are provided by the host processor, and the output signal is the serial data stream. The architecture of the transmitter consists of a control unit, a data register (XMT_datareg), a data shift register (XMT_shfreg), and a status register (bit_count), which counts the bits that are transmitted. The status register will be included with the datapath unit.

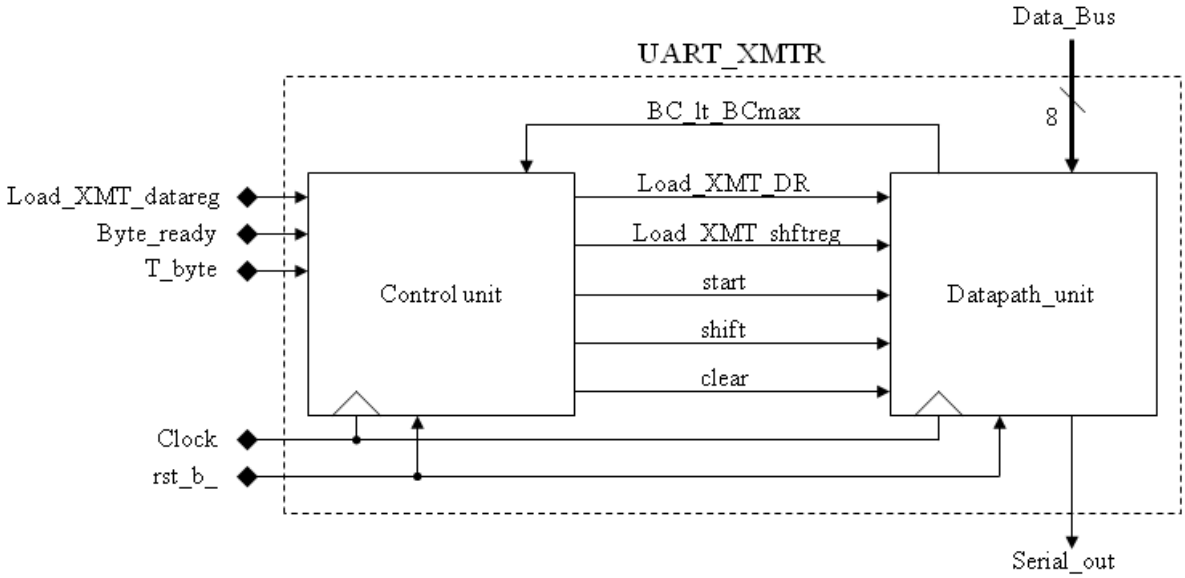


Figure 5 Interface signals for the logic of a state machine controller for a UART transmitter

The controller has the inputs (primary/external and status (from the datapath)) listed below. We note that the signal *Load_XMT_datareg* could be passed directly to the datapath unit; instead, we pass *Load_XMT_datareg* to the control unit and assert *Load_XMT_DR* conditionally when the state is *idle* and the external signal *Load_XMT_datareg* is asserted. The status signal, *BC_lt_BCmax*, is asserted while bits are being sent, i.e., if *Bit_count* < *word_size* +1.

a. *Load_XMT_datareg*: assertion is state idle asserts *Load_XMT_DR*, which loads the content of the *Data_Bus* into *XMT_data_reg*

b. *Byte_ready*: assertion causes *Load_XMT_shfreg* to assert, which loads the contents of *XMT_datareg* into *XMT_shfreg*

c. *T_byte* : assertion initiates transmission of a byte of data, including the stop, start, and parity bits

d. *BC_lt_BCmax*: indicates status of the bit counter in the datapath unit

The state machine of the controller forms the following output signals that control the datapath of the transmitter:

a. *Load_XMT_DR*: assertion loads *Data_Bus* into *XMT_datareg*

b. Load_XMT_shfreg: assertion loads the contents of *XMT_data_reg* into *XMT_shftreg*

c. *start*: signals the start of transmission by dropping *XMT_shftreg*[0] to 0

d. *shift*: directs *XMT_shftreg* to shift by one bit towards the LSB and to backfill with a stop bit(1).

e. *clear* : clears bit_count

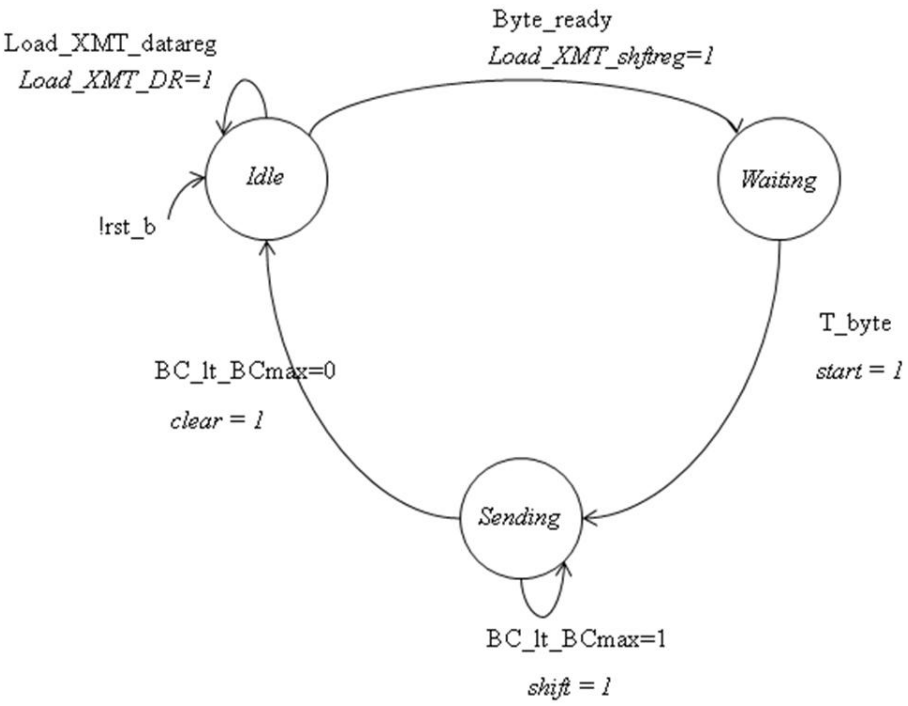
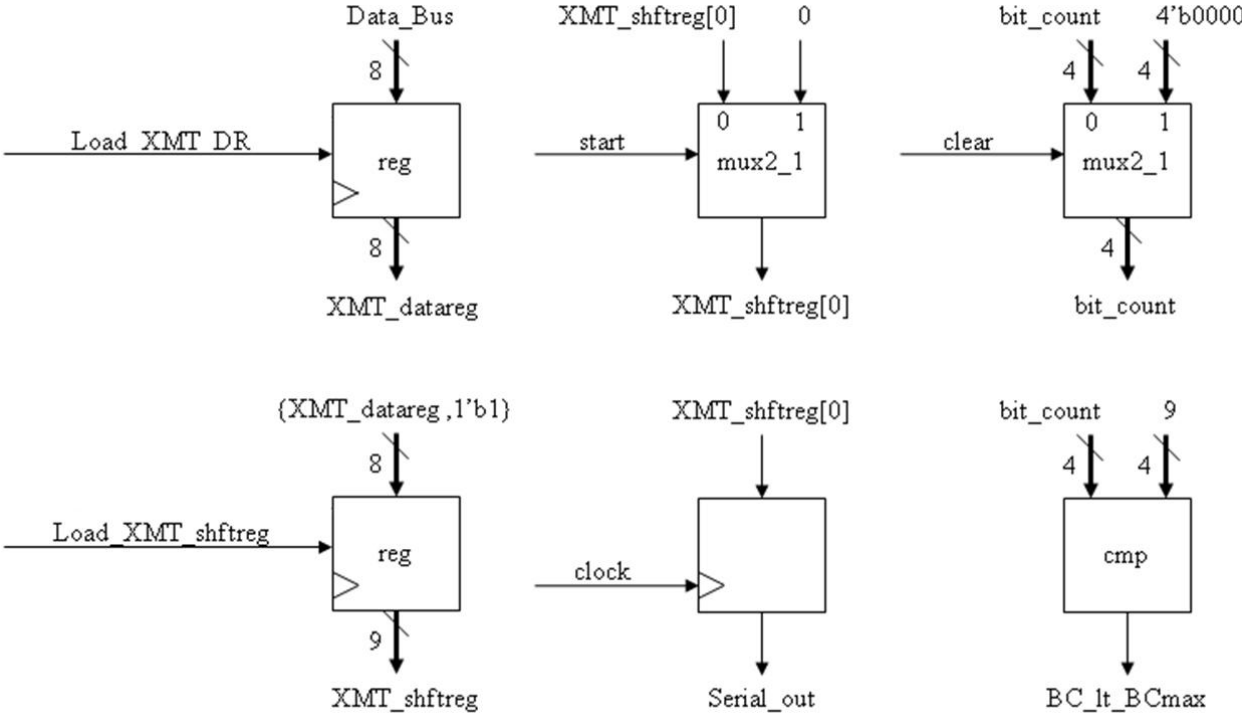


Figure 6 High-Level State Machine (HLSM) of the control unit



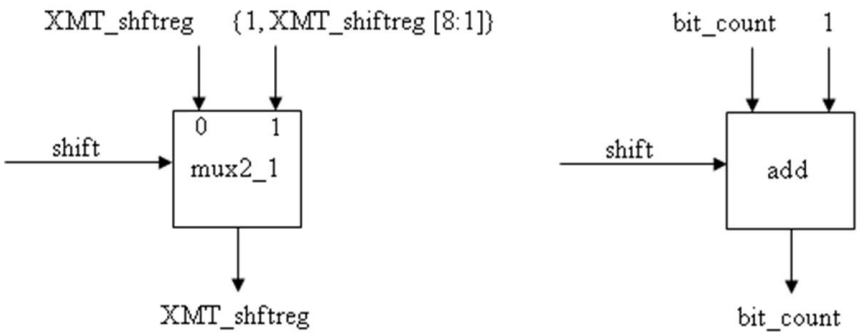


Figure 7 Datapath unit of the transmitter

The HLSM of the control unit and datapath unit of the transmitter is shown in Figure 6 and Figure 7. The machine has three states: *idle*, *waiting*, and *sending*. When the active-low, synchronous reset signal *rst_b* is asserted, the machine enters *idle*, *bit_count* is flushed, and *XMT_shftreg* is loaded with 1s. In *idle*, if an active edge of *Clock* occurs while *Load_XMT_data_reg* is asserted by the external host, the output signal *Load_XMT_DR* will load *XMT_data_reg* with the contents of *Data_Bus*. The machine remains in *idle* until start is asserted to drop *XMT_shftreg*[0].

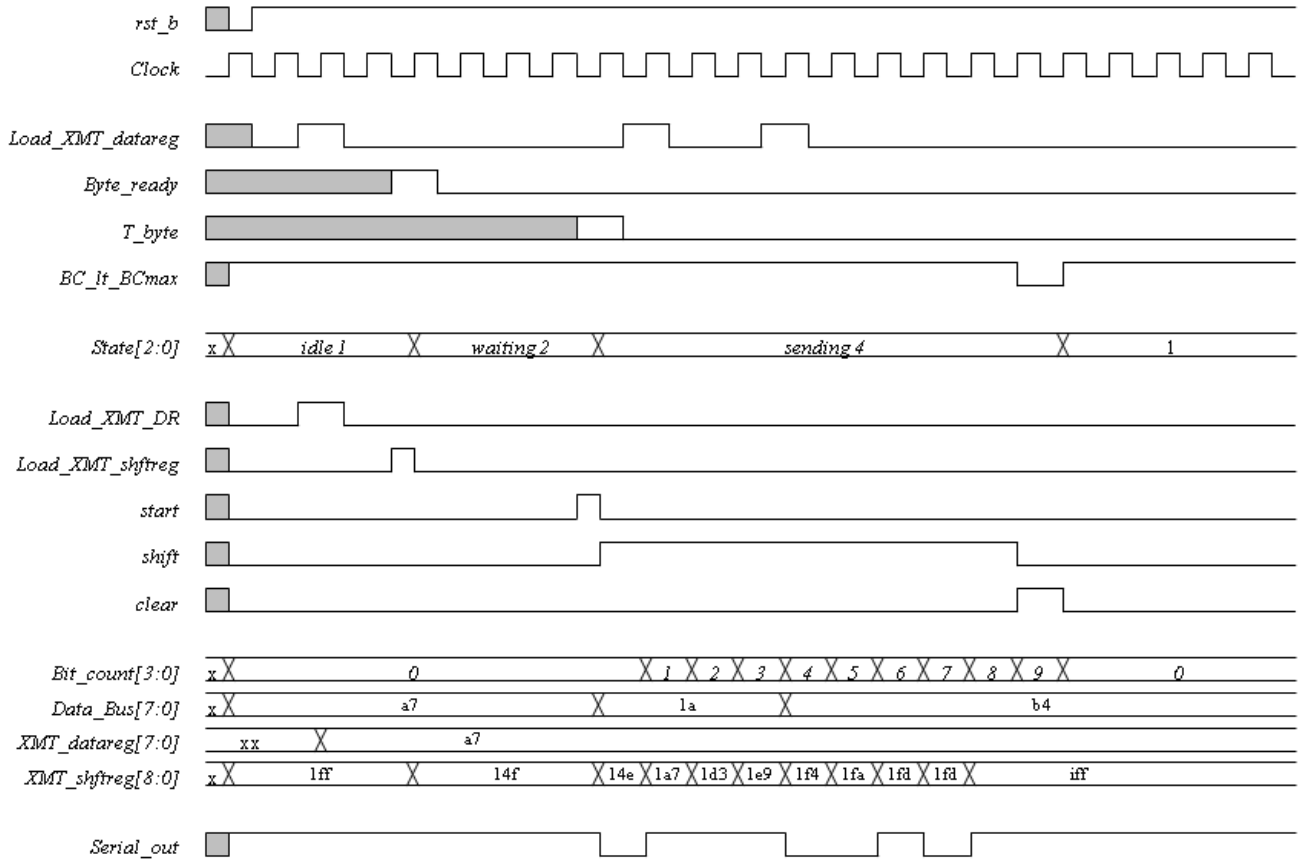


Figure 8 Timing simulation results for the 8-bit UART transmitter

Figure 8 shows an example of timing of transmission. You can design your testbench referring this timing graph. Also, you can check your result based on this timing graph.

3. User Defined Primitives (UDP)

Verilog has built-in primitives like gates, transmission gates, and switches. This is a rather small number of primitives, if we need more complex primitives, then Verilog provides UDP, or simply User Defined Primitives. Using UDP we can model combinational logic and sequential logic. We can include timing information along with these UDP to model complete ASIC library models. **In this lab, we will design comparing block to generate signal *BC_lt_BCmax* using UDP. The signal BC_lt_BCmax should be '1' if bit_count is less than 9 which is the maximum number of bits transmitted, otherwise '0'. Please put output port as a first variable.**

Example of UDP

```
// This code shows how UDP body looks like
primitive UDP_NAME (
        a, // Port a
        b, // Port b
        c, // Port c
);
// In/Out declaration
output a;
input b,c;

// UDP function code here
// A = B | C;
table
        // B C : A
        ?   1   :   1;
        1   ?   :   1;
        0   0   :   0;
endtable
endprimitive
```

4. Functional Simulation

Once the design is completed it must be tested. The functionality of the design module can be tested by applying a testbench and checking the results. The testbench module can instantiate the design module and directly drive the signals in the design module. The testbench can be complied along with the design module. At the end of compilation the simulation results will be displayed.

Download 'Lab8_code.tar.gz' file from the lecture website. The file contains UART_XMTR.v, Control_Unit.v, Data_Path.v, UART_tb.v, generic.sdb, osu018_stdcells.v and osu018_stdcells.db. You will design modules in these files after you decompress the file.

a. Make working folder for this lab.

HOME]$ mkdir ECEN468/Lab8/SRC (make work folder)
HOME]$ cd ECEN468/Lab8/SRC (move to work folder)
Move the file 'Lab8_code.tar.gz' into the working folder
ECEN468/Lab8/SRC]$ tar xvfz Lab8_code.tar.gz (decompress the file)

Please check the files decompressed. It should contain the files below.
UART_XMTR.v, Control_Unit.v, Data_Path.v, UART_tb.v, generic.sdb
osu018_stdcells.v, osu018_stdcells.db

b. Insert your code into module files (UART_XMTR.v, Control_Unit.v, Data_Path.v).

d. Start Functional Simulation

ECEN468/Lab8/SRC]$ vcs UART_tb.v

If it shows compile errors, please check your codes again (go to step b or c). If compilation is complete, go to next step to get the results.

ECEN468/Lab8/SRC]$ simv

If it is not same to your estimated results, please check your codes again (go to step b or c). Please refer to step e for your debug process. **If the result is correct, copy the output results and save them into new file. You will need to submit the file.** Please show your result to TA within your regular lab hour to get credit for lab evaluation.

e. How to debug

To view graphical waveform of signals through WaveView, use this code in your testbench. Also you can use $monitor or $display syntax to print messages in your codes.

```
initial
begin
        $dumpfile ("wave.dump");
        $dumpvars(0, stimulus);
end
```

5. Starting design analyzer

To start design analyzer, please follow steps. For your convenience, please start *design_analyzer* from separate directory with verilog files(*.v) because it generates a lot of intermediate files while it goes further.

Make directory for design analyzer. Now, you should be in ECEN468/Lab8/SRC.

ECEN468/Lab8/SRC]$ mkdir DV_WORK	(make work folder)
ECEN468/Lab8/SRC]$ cd DV_WORK	(move to the work folder)
ECEN468/Lab8/SRC/DV_WORK]$ design_vision &	(execute Design Analyzer)

6. Procedures of synthesis process:

6.1. Procedure to link the libraries given with the synthesis process:

a. After you begin *Design_Analyzer* , choose "*Setup-->Defaults*". A window pops up.
b. Fill in *osu018_stdcells.db* in "*Link Library*". (Delete all other library files)
c. Fill in *osu018_stdcells.db* in "*Target Library*". (Delete all other library files)
d. Fill in *generic.sdb* in the "*Symbol Library*" (Delete all other library files)
e. Click *Ok.*

6.2. Load a design

Analyze, Elaborate are commands provided by Design Compiler. They are available through the *File* menu in *Design Analyzer*.

The *Analyze* command reads a VHDL or Verilog file, checks for proper syntax and synthesizable logic, and stores the design in an intermediate format.

The *Elaborate* command creates a design from the intermediate format produced by *Analyze*. It replaces the HDL operators with synthetic operators and determines the correct bus sizes.

In this lab, *Analyze* and *Elaborate* are used to load the files.

6.3. Detailed procedures:

a) Analyze File

Select *"File->Analyze"*.

The Analyze File window appears as shown in Figure 9. Change the Format to **Auto.**

Click Add button, and select *UART_XMTR.v* (Top module).

Work library to **WORK**. Click *Ok*.

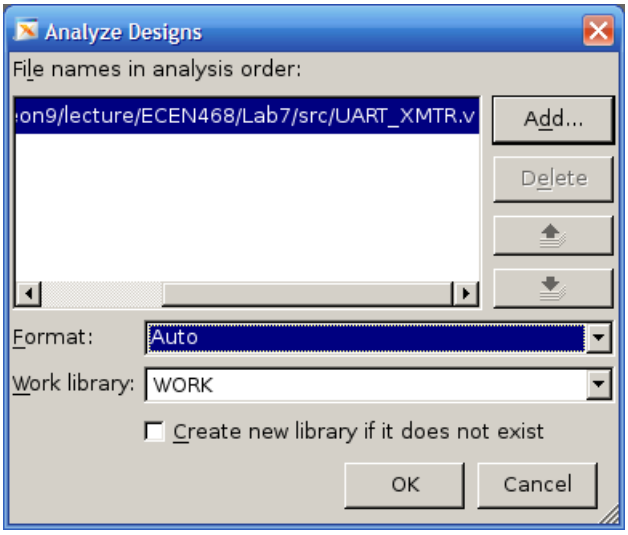


Figure 9 Analyze File window

You will meet a problem about UDP designed while you analyze the file. It is not synthesizable, so you need to use the line below instead of UDP.

assign BC_lt_BCmax = (bit_count < word_size + 1);

b) Elaborate Design

Select *"File->Elaborate"*. The Elaborate Design window appears.

Choose the *work* library in the Library list.

Select the *UART_XMTR* (Top module) in the Design list. Click *Ok*.

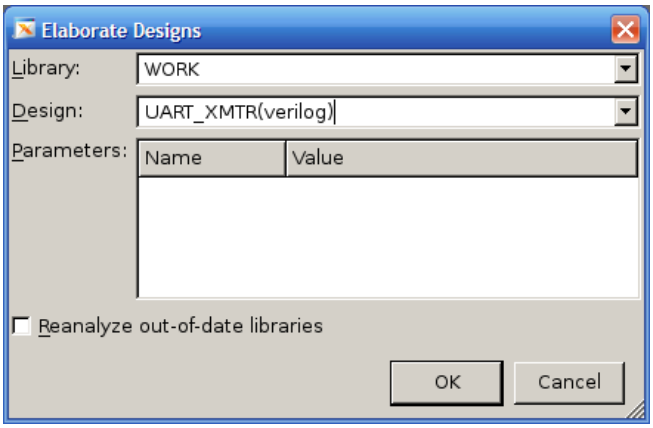


Figure 10 Elaborate Design window

c) Synthesize the modules

Use ***Design-Compile Design***. And Click *OK*.

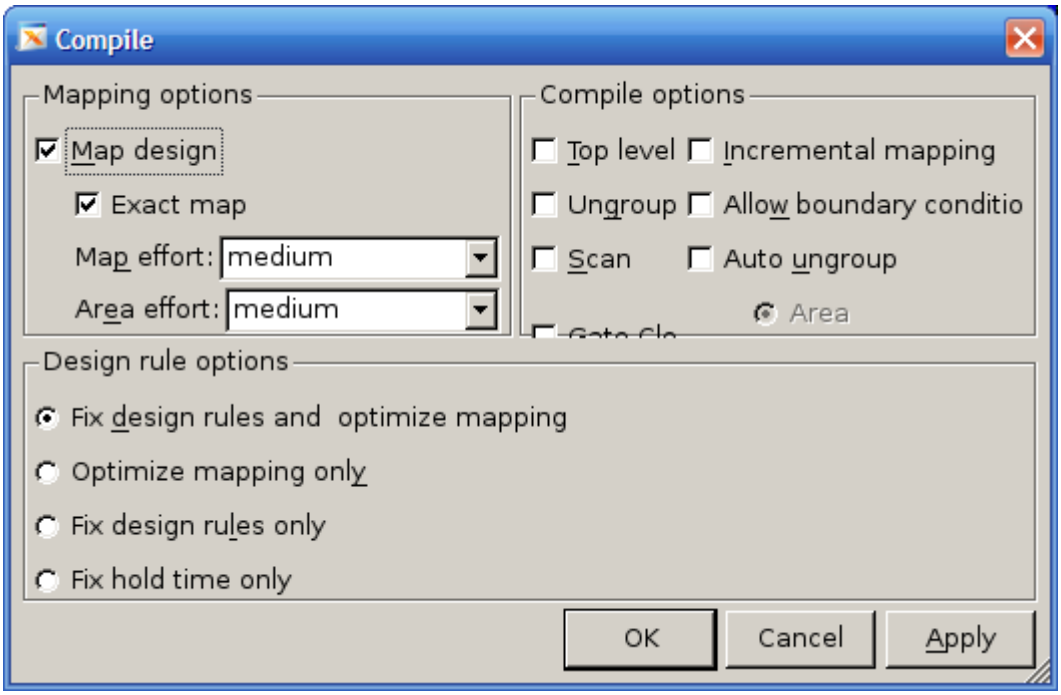


Figure 12 Compile Window

d) Save the optimized verilog netlist.

Select *TOP* design (UART_XMTR)

Select *"File->Save As"*.

Enter ***UART_gate.v*** in the File Name field to save the file.

Choose the **VERILOG (v)** option in Format list.

Check the *Save All Designs in Hierarchy* option.

Make sure you save the synthesized netlist of the design in verilog format.

e) Save the Standard Delay Format (SDF).

Save the design in the form a Standard Delay Format.

Use command "write_sdf UART.sdf" in command window of Design Analyzer.

7. Gate Simulation

a) Copy Verilog netlist and sdf file to work folder.

ECEN468/Lab8/SRC/DV_WORK]$ cp UART_gate.v **../.** (copy it to work folder)
ECEN468/Lab8/SRC/DV_WORK]$ cp UART.sdf **../.** (copy it to work folder)
ECEN468/Lab8/SRC/DV_WORK]$ cd **..** (move to work folder)

b) Follow steps below to make your testbench for gate simulation
Now we will make testbench for gate simulation. For gate simulation, we need UART_gate.v (netlist), UART.sdf (SDF file), osu018_stdcells.v and testbench.

Please make sure those files should be in the current working folder. If not, please let TA know.

ECEN468/Lab8/SRC]$ cp UART_tb.v UART_gate_tb.v (copy new testbench)
Please check that the below lines are in UART_gate_tb.v

On top of the file,

```
`timescale 1ns/10ps
`include "UART_gate.v"
`include "osu018_stdcells.v"
```

On the bottom of the file, **(This line should be inside module)**

```
Initial
        $sdf_annotate("UART.sdf", UART_XMTR_01);
```

Delete the line `include "UART_XMTR.v" in your testbench
Change the name of dump file to "wave_gate.dump" in your testbench

c) Start gate simulation

ECEN468/Lab8/SRC]$ vcs UART_gate_tb.v

If there is no syntax error, then do next step.

ECEN468/Lab8/SRC]$ simv

Then, you will see the text message and get wave_gate.dump file. We sometimes want to see the waveform to debug. Please compare the result with the functional simulation result. If it is not same to your estimation, start simulation again from the functional simulation after you modify design modules. **Also, you may try decreasing 2 or 3 times slower frequency in the test-bench given.**

Please show your result to TA within your regular lab hour to get credit for lab evaluation.

Requirements:

1. General requirements.
   a. It should control UART correctly.
      - Protocol : Start bit('0')+Data bits(8bits)+Stop bit('1')
      - Input Ports : Load_XMT_datareg, Byte_ready, T_byte, rst_b, Data_Bus, Clock
      - Output Port : Serial_out
      - Design an UDP block for the signal *BC_lt_BCmax* in the functional simulation
   b. Detailed comments.
   c. Please do not change the names of the signals given and the other functions not mentioned in the report. (You will lose some points if you change any names of the input or output signals.)
   d. You may modify the test-bench given to get the results only if it is resaonable. Please write detailed comments if you have modified.
   e. Late penalty : 20% of total score will be deducted on each subsequent weekday after due date.

2. Submit hardcopy of your report to TA. The report should include contents below
   a. Text result of the functional simulation (3 points)
   b. Waveform result(screen captured) of the function simulation with analysis. (5 points)
   c. Text result of the gate simulation (3 points)
   d. Waveform result(screen captured) of the gate simulation with analysis. (5 points)
   e. UART_XMTR.v, Control_Unit.v and Datapath_Unit.v with detailed comments (4 points)
   f. Only the parts modified in UART_tb.v and UART_gate_tb.v, if you have modified. (2 points)
   g. UDP block for the signal *BC_lt_BCmax* (8 points)
   h. Calculate the maximum clock speed of the test-bench you used if you have changed clock period of the testbench. Otherwise, calculate the frequency given in the testbench. (5 points) (You will get scores based on the maximum clock speed.)

Note : Please send these files to TA's email address. (5 points)

UART_XMTR.v Control_Unit.v, Datapath_Unit.v, UART_tb.v, URAT_gate_tb.v, wave.dump, wave_gate.dump, UART_gate.v, UART.sdf, text messages(functional simulation and gate simulation)
